12-14-2001

Accurate Local Time Stepping Schemes for Non-Linear Partial Differential Equations

Kiran Kumar V Adhikarala

ACCURATE LOCAL TIME STEPPING SCHEMES FOR NON-LINEAR PARTIAL DIFFERENTIAL EQUATIONS

By

Kiran Kumar V. Adhikarala

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master Of Science
in Computational Engineering
in the College of Engineering

Mississippi State, Mississippi

December 2001

ACCURATE LOCAL TIME STEPPING SCHEMES FOR NON-LINEAR PARTIAL DIFFERENTIAL EQUATIONS

By

Kiran Kumar V. Adhikarala

Approved:

Edward A. Luke Assistant Professor of Computer Science (Director of Thesis)	Pasquale Cinnella Associate Professor of Aerospace Engineering (Committee Member)
Bharat K. Soni Professor of Aerospace Engineering (Committee Member)	Boyd Gatlin Associate Professor of Aerospace Engineering and Engineering Mechanics (Graduate Coordinator)

A. Wayne Bennett
Dean of the College of Engineering

Name: Kiran Kumar V. Adhikarala

Date of Degree: December 14, 2001

Institution: Mississippi State University

Major Field: Computational Engineering

Major Professor: Dr. Edward A. Luke

Title of Study: ACCURATE LOCAL TIME STEPPING SCHEMES FOR NON-LINEAR PARTIAL DIFFERENTIAL EQUATIONS

Pages in Study: 67

Candidate for Degree of Master Of Science

This study seeks to reduce the cost of numerically solving non-linear partial differential equations by reducing the number of computations without compromising accuracy. This was done by using accurate local time stepping. This algorithm uses local time stepping but compensates for the inconsistencies in the temporal dimension by interpolations and/or extrapolations. Reduction in computations are obtained by time-stepping only a particular region with small time steps. A shock tube problem and a detonation wave were the two test cases considered. The performance of the solution using this algorithm was compared with an algorithm that does not use accurate local time stepping.

DEDICATION

I would like to dedicate this research to my 'mamma gaaru' and my parents.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my major professor, Dr. Edward Luke, for his valuable guidance and assistance. I would also like to say a special "thank you" to Dr. Jianping Zhu, without whose constant help and encouragement, I would not have been able to acomplish the whole task. I also would like to thank Dr. Pasquale Cinnella and Dr. Bharat Soni for their contributions. In addition, I would like to thank Dr. Boyd Gatlin and the Engineering Research Center for providing financial support and facilities to this research effort.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

NOMENCLATURE

Fluid Dynamic Variables:

$\mathcal{A}_f$	Area of a Face
$c_{v\,s}$	Species Specific Heat at Constant Volume
e_0	Fluid Total Energy
$h_{f\,s}$	Species Heat of Formation
$K_{c,r}$	Reaction Equilibrium Constant
$k_{f,r}$	Forward Reaction Rate
$k_{b,r}$	Backward Reaction Rate
k	Fluid Thermal Conductivity
I	Identity Matrix
$\tilde{\tilde{I}}$	Identity Tensor
$\mathcal{M}_s$	Species Atomic Mass
p	Fluid Pressure
p_{ref}	Reference Pressure
$\hat{R}$	Universal Gas Constant
R_s	Gas Constant for species s
s_{ref}	Entropy measured at Reference conditions

T	Fluid Temperature
T_{ref}	Reference Temperature
t	Time
$\tilde{u}$	Fluid Velocity Vector
$\tilde{u}_\Omega$	Grid Velocity Vector
$\mathcal{V}_c$	Volume of a Cell
$\dot{w}_s$	Species Production Rate
Y_s	Species Mass Fraction (ρ_s/ρ)
Γ	Surface of Space Domain
θ_v	Characteristic Vibrational Temperature
ρ	Fluid Density
ρ_s	Species Density
$\tilde{\tilde{\sigma}}$	Deviatoric Stress Tensor
Ω	Volume of Space Domain
Tensor Operators:	
curl	Curl
div	Divergence
grad	Gradient
Logical Operators:	
$\neg$	Logical Negation
$\Rightarrow$	Logical Implication
$\rightarrow$	Mapping (composition) operator

$a \leftarrow b$	Rule operator, meaning b generates a
$\oplus$	A generic operator

CHAPTER I

INTRODUCTION

The solution of many engineering problems involves the solution to some set of partial differential equations. For example, engineering problems involving fluid flows may require the solution to the Navier-Stokes equations. For almost all engineering applications, obtaining analytical solutions for these PDE might not only be impractical, but might also be impossible, mostly because of the non-linearity of the equations and also because of the complex geometries involved. Thus, usually, approximate numerical solutions are sought, using some algorithms like a finite volume scheme. In case the solution needed to be more accurate in some portions of the grid than others, we would be left with no choice except that we had to reduce the entire grid size or do more iterations for every time step. Instead, we can do what is called accurate local time stepping. Local time stepping has been used to improve the computational efficiency of steady state solutions. The idea of the present study is to apply the basic concept of local time stepping for the calculation of unsteady solutions without compromising accuracy.

The basic idea of local time stepping is that the different grid points are temporally advanced in different time steps depending on the stable CFL number [1, 2, 3, 4]. For steady state solution (where time derivatives vanish), it would

not matter whether the solution is advanced in different time steps, because we are interested in the steady state solution which should become independant of the time step after a certain amount of time. This property is exploited and thus local time stepping expedites the rate of convergence. Evidently, traditional local time stepping can not be used for obtaining unsteady solutions. This is because time consistency is not satisfied [5]. In order to use local time stepping for unsteady solutions, we need to ensure that time consistency is maintained. This can be accomplished through interpolations and extrapolations to match the solutions calculated at different time steps, and to advance solutions at all spatial grid points to the same time level, eventually, by using sub-step iterations.

We could use an explicit scheme in each time step and the corresponding sub-steps, or, we could use implicit methods to advance in time. In [6, 7], an accurate local time stepping algorithm based on explicit time integration methods was developed for calculating unsteady solutions using interpolations. For many applications, the explicit scheme might not be efficient because of the severe instability problems. For example, in chemical reactive flow simulations [8], the reaction term in the equations could be very stiff, making explicit algorithms virtually impractical for solving large scale problems. Therefore, it is desirable to develop accurate local time stepping algorithms using implicit time integration methods for calculating unsteady solutions. In the present work, due to the non-linear nature of the Navier-Stokes equations, and the above mentioned reason, the implicit time integration method has been used.

We introduce the concept of accurate local time stepping i.e., how it is done, with the help of an example.To simplify the discussion, we consider the following simple model equation:

$$u_t + u_x = 0, \qquad t > 0, \qquad x \in (0,1), \tag{1.1}$$

$$u(0,t) = g(t), \qquad t > 0,$$

$$u(x,0) = f(x), \qquad x \in [0,1].$$

The method and the analysis are applicable to more general cases involving system of nonlinear equations in higher dimensional spatial domains. Discretizing Eq.

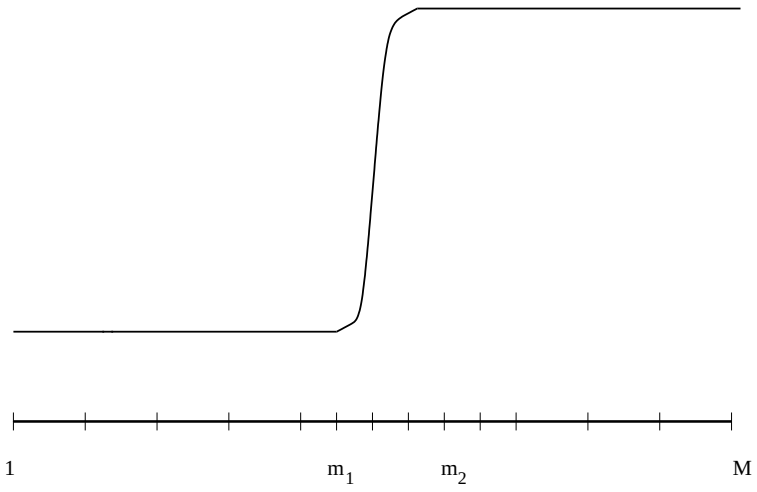

Figure 1.1: A Solution that has a steep gradient in the middle

(1.1) using backward time and first order upwind finite difference, we have

$$\frac{u_j^{n+1} - u_j^n}{\mathrm{dt}} + \frac{u_j^{n+1} - u_{j-1}^{n+1}}{\mathrm{dx}} = 0, \qquad j = 2, \cdots, M, \tag{1.2}$$

$$u_1^{n+1} = g^{n+1}, \qquad u_j^0 = f_j, \qquad j = 1, \cdots, M,$$

or

$$(1+c)u_j^{n+1} - cu_{j-1}^{n+1} = u_j^n, \qquad j = 2, \cdots, M, \tag{1.3}$$

$$u_1^{n+1} = g^{n+1}, \qquad u_j^0 = f_j, \qquad j = 1, \cdots, M,$$

where dt and dx are the step sizes in the temporal and spatial dimensions, respectively, M is the total number of spatial grid points, and $c = \frac{\mathrm{dt}}{\mathrm{dx}}$ is the local CFL number. This can also be written in matrix form:

$$\begin{bmatrix} 1+c & & & & \\ -c & 1+c & & & \\ & \ddots & \ddots & & \\ & & -c & 1+c & \\ & & & -c & 1+c \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \\ \vdots \\ u_{M-1} \\ u_M \end{bmatrix}^{n+1} = \begin{bmatrix} u_2 + cg^{n+1} \\ u_3 \\ \vdots \\ u_{M-1} \\ u_M \end{bmatrix}^{n}. \tag{1.4}$$

Although the coefficient matrix in (1.4) is bi-diagonal, it will be tri-diagonal (or block tri-diagonal) in general when central differences are used, or when a system of equations with different characteristic directions is being solved.

For many application problems, the solutions have steep gradients in only a very small part of the spatial domain. A non-uniform grid is usually employed in these cases for better computational accuracy. Fig. 1.1 shows a one-dimensional example in which the solution changes rapidly in the middle of the spatial domain, where the grid points are more densely distributed. When the computation is

carried out using a uniform CFL number $c = \frac{\mathrm{dt}}{\mathrm{dx}} = constant$ for all grid points, the solutions at all grid points are not advanced to the same time level after each step. As shown in Fig. 1.2, if dx_1=2dx_2, and $\mathrm{dt}_1 = 2\mathrm{dt}_2$, with a constant CFL number $c = \frac{\mathrm{dt}_1}{\mathrm{dx}_1} = \frac{\mathrm{dt}_2}{\mathrm{dx}_2}$, then the solutions $u_2, \cdots, u_{m_1-1}$ and $u_{m_2+1}, \cdots, u_M$ at the two sides of the spatial domain are advanced twice as fast as the solutions $v_{m_1}, \cdots, v_{m_2}$ in the middle. Apparently, this mis-match in time levels causes the loss of temporal accuracy, which makes the traditional local time stepping algorithms not suitable for calculating accurate unsteady solutions.

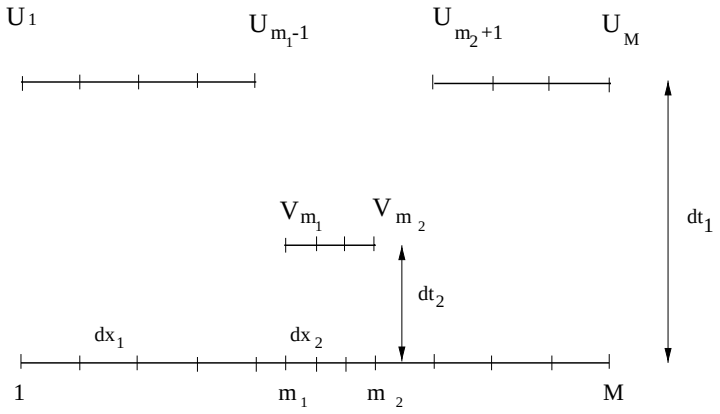

Figure 1.2: Non-uniform advancement in the temporal dimension.

1.1 Accurate Implicit Local Time Stepping Algorithms

To see how the traditional local time stepping algorithm affects the temporal accuracy of the calculations, we first look at the finite difference equation at the grid point m_1 shown in Fig. 1.3:

$$\frac{\tilde{v}_{m_1}^{n+1} - v_{m_1}^{n}}{\mathrm{dt}_2} + \frac{\tilde{v}_{m_1}^{n+1} - \tilde{v}_{m_1-1}^{n+1}}{\mathrm{dx}_2} = 0, \tag{1.5}$$

where m_1 is the first grid point at which the smaller grid spacing dx_2 is used in the calculation. Although the finite difference method is used here to discretize Eq. (1.1), the basic idea of the new local time stepping algorithm discussed in this thesis also applies to other discretization methods, such as finite volume and finite element methods.

Note that the value $\tilde{v}_{m_1-1}^{n+1}$ in Eq. (1.5) is undefined. What is used in the traditional local time stepping algorithm is actually the value of $u_{m_1-1}^{n+1}$. Similarly,

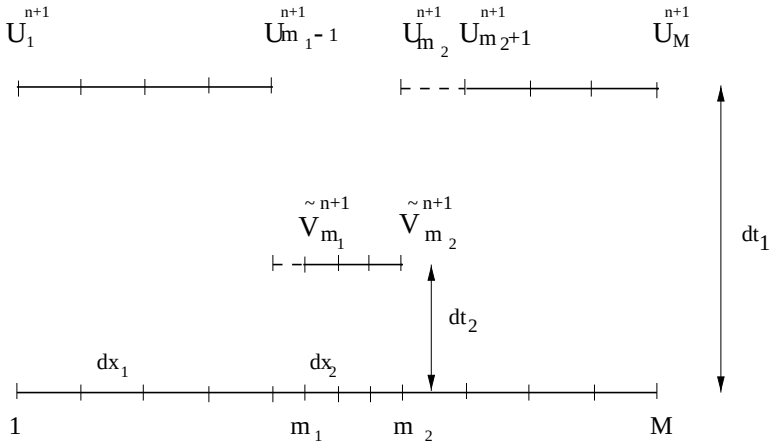

Figure 1.3: Usage of solutions in different time levels in the same equation

at the grid point $m_2 + 1$, we have

$$\frac{u_{m_2+1}^{n+1} - u_{m_2+1}^{n}}{\mathrm{dt}_1} + \frac{u_{m_2+1}^{n+1} - u_{m_2}^{n+1}}{\mathrm{dx}_1} = 0, \tag{1.6}$$

where the point $m_2 + 1$ is the transition point from small grid spacing dx_2 to large grid spacing dx_1. Similar to the equation at the grid point m_1, the value $u_{m_2}^{n+1}$ is not defined. What is used in the traditional local time stepping algorithm is

actually the value $\tilde{v}_{m_2}^{n+1}$. It is now clear that the accuracy of traditional local time stepping algorithms is affected by

- using solutions defined at different time levels in the same discretization formula, such as $u_{m_1-1}^{n+1}$ in Eq. (1.5) and $\tilde{v}_{m_2}^{n+1}$ in Eq. (1.6), and
- advancing solutions to different time levels using different time steps. It is clear from Fig. 1.3 that after one time step, the solutions $u_i^{n+1}, i = 2, \cdots, m_1 - 1, m_2 + 1, \cdots, M$ are not defined at the same time level as the solutions $\tilde{v}_i^{n+1}, i = m_1, \cdots, m_2$.

To address the first problem, we use the first order interpolation

$$\tilde{v}_{m_1-1}^{n+1} = \frac{1}{2}(u_{m_1-1}^{n+1} + u_{m_1-1}^{n}) + O(\mathrm{dt}_2^2) \tag{1.7}$$

to approximate $\tilde{v}_{m_1-1}^{n+1}$, and use the first order extrapolation

$$u_{m_2}^{n+1} = 2\tilde{v}_{m_2}^{n+1} - v_{m_2}^{n} + O(\mathrm{dt}_2^2) \tag{1.8}$$

to approximate $u_{m_2}^{n+1}$.

With the approximations given in (1.7) and (1.8), we can write Eqs. (1.5) and (1.6) as

$$\frac{\tilde{v}_{m_1}^{n+1} - v_{m_1}^{n}}{\mathrm{dt}_2} + \frac{\tilde{v}_{m_1}^{n+1} - \frac{1}{2}(u_{m_1-1}^{n+1} + u_{m_1-1}^{n})}{\mathrm{dx}_2} = 0, \tag{1.9}$$

and

$$\frac{u_{m_2+1}^{n+1} - u_{m_2+1}^{n}}{\mathrm{dt}_1} + \frac{u_{m_2+1}^{n+1} - (2\tilde{v}_{m_2}^{n+1} - v_{m_2}^{n})}{\mathrm{dx}_1} = 0, \tag{1.10}$$

respectively, which leads to the following matrix equation system:

$$\begin{bmatrix}
\sigma & & & & & & & & & \\
-c & \sigma & & & & & & & & \\
 & \ddots & \ddots & & & & & & & \\
 & & -c & \sigma & & & & & & \\
 & & & -\frac{c}{2} & \sigma & & & & & \\
 & & & & \ddots & \ddots & & & & \\
 & & & & & -c & \sigma & & & \\
 & & & & & & -2c & \sigma & & \\
 & & & & & & & \ddots & \ddots & \\
 & & & & & & & & -c & \sigma
\end{bmatrix}
\begin{bmatrix} u_2 \\ u_3 \\ \vdots \\ u_{m_1-1} \\ \tilde{v}_{m_1} \\ \vdots \\ \tilde{v}_{m_2} \\ u_{m_2+1} \\ \vdots \\ u_M \end{bmatrix}^{n+1}
=
\begin{bmatrix} u_2 + cg^{n+1} \\ u_3 \\ \vdots \\ u_{m_1-1} \\ v_{m_1} + \frac{c}{2}u_{m_1-1} \\ \vdots \\ v_{m_2} \\ u_{m_2+1} - cv_{m_2} \\ \vdots \\ u_M \end{bmatrix}^{n}, \tag{1.11}$$

where $\sigma = 1 + c$. Comparing Eq. (1.11) with Eq. (1.4), we can see that the only differences occur in the two finite difference equations at the grid points m_1 and $m_2 + 1$, respectively.

To address the problem that solutions at different spatial grid points are advanced to different time levels due to the difference in time step sizes, we introduce sub-step iterations to bring all solutions to the same time level. As shown in Fig. 1.4, assume $\mathrm{dt}_1 = 2\mathrm{dt}_2$. The solutions at all spatial grid points can be advanced to the same time level in two sub-steps. In the first sub-step of the calculation, solutions

$$\{u_2^{n+1}, \cdots, u_{m_1-1}^{n+1}, \tilde{v}_{m_1}^{n+1}, \cdots, \tilde{v}_{m_2}^{n+1}, u_{m_2+1}^{n+1}, \cdots, u_M^{n+1}\}^T$$

are calculated. Note that after this step, the values of $\tilde{v}_{m_1}^{n+1}, \cdots, \tilde{v}_{m_2}^{n+1}$ are defined at the time level $t_n + \mathrm{dt}_2$, while all other solutions are defined at $t_n + \mathrm{dt}_1$.

In the second sub-step, the solutions

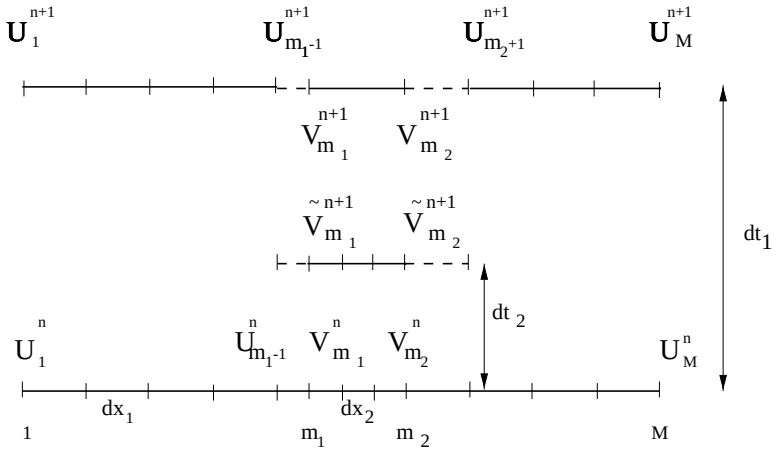

Figure 1.4: Usage of sub-steps for compensating temporal discrepancies

$$\{v_{m_1}^{n+1}, \cdots, v_{m_2}^{n+1}\}^T \tag{1.12}$$

are calculated using $u_{m_1-1}^{n+1}$ as the boundary condition. Specifically, the equation at the grid point m_1 is

$$\frac{v_{m_1}^{n+1} - \tilde{v}_{m_1}^{n+1}}{\mathrm{dt}_2} + \frac{v_{m_1}^{n+1} - u_{m_1-1}^{n+1}}{\mathrm{dx}_2} = 0, \tag{1.13}$$

and all other equations for the grid points $m_1 + 1, \cdots, m_2$ are

$$\frac{v_j^{n+1} - \tilde{v}_j^{n+1}}{\mathrm{dt}_2} + \frac{v_j^{n+1} - v_{j-1}^{n+1}}{\mathrm{dx}_2} = 0, \qquad j = m_1 + 1, \cdots, m_2, \tag{1.14}$$

which leads to the following system of equations for the second sub-step

$$\begin{bmatrix} 1+c & & & & \\ -c & 1+c & & & \\ & \ddots & \ddots & & \\ & & -c & 1+c & \\ & & & -c & 1+c \end{bmatrix} \begin{bmatrix} v_{m_1} \\ v_{m_1+1} \\ \vdots \\ v_{m_2-1} \\ v_{m_2} \end{bmatrix}^{n+1} = \begin{bmatrix} \tilde{v}_{m_1} + cu_{m_1-1} \\ \tilde{v}_{m_1+1} \\ \vdots \\ \tilde{v}_{m_2-1} \\ \tilde{v}_{m_2} \end{bmatrix}^{n+1}. \tag{1.15}$$

In general, the solution value $u_{m_2+1}^{n+1}$ may also be needed as boundary conditions, depending on the characteristic directions and the discretization scheme used in the calculation. If $\mathrm{dt}_2 = \frac{\mathrm{dt}_1}{L}$, then the solution vector $\{v_{m_1}^{n+1}, \cdots, v_{m_2}^{n+1}\}^T$ will be calculated using L sub-steps. In the first substep, the solution vector $\{\tilde{v}_{m_1}^{n+\frac{1}{L}}, \cdots, \tilde{v}_{m_2}^{n+\frac{1}{L}}\}^T$ is calculated together with $\{u_2^{n+1}, \cdots, u_{m_1-1}^{n+1}\}^T$ and $\{u_{m_2+1}^{n+1}, \cdots, u_M^{n+1}\}^T$ by solving Eq. (1.11). Note that the interpolation and extrapolation formulas given in Eqs. (1.7) and (1.8) should be modified as

$$\tilde{v}_{m_1-1}^{n+\frac{1}{L}} = \frac{1}{L} u_{m_1-1}^{n+1} + \frac{L-1}{L} u_{m_1-1}^{n} + O(\mathrm{dt}_2^2), \tag{1.16}$$

and

$$u_{m_2}^{n+1} = L\tilde{v}_{m_2}^{n+\frac{1}{L}} - (L-1)v_{m_2}^{n} + O(\mathrm{dt}_2^2), \tag{1.17}$$

respectively, which requires changing the number 2 in Eq. (1.11) to L and the value $-c$ in the right hand side of Eq. (1.11) to $(1-L)c$, respectively. In the last sub-step, the solution vector $\{v_{m_1}^{n+1}, \cdots, v_{m_2}^{n+1}\}^T$ is calculated by solving a system

of equations similar to Eq. (1.15). For the $L-2$ sub-steps in between, we need to use the following interpolation formula

$$\tilde{v}_{m_1-1}^{n+\frac{q}{L}} = \frac{q}{L} u_{m_1-1}^{n+1} + \frac{L-q}{L} u_{m_1-1}^{n} + O(\mathrm{dt}_2^2) \tag{1.18}$$

to provide the boundary condition for calculating $\{\tilde{v}_{m_1}^{n+\frac{q}{L}}, \cdots, \tilde{v}_{m_2}^{n+\frac{q}{L}}\}^T$, $q = 2, \cdots, L-1$. The corresponding system of matrix equations is

$$\begin{bmatrix} 1+c & & & & \\ -c & 1+c & & & \\ & \ddots & \ddots & & \\ & & -c & 1+c & \\ & & & -c & 1+c \end{bmatrix} \begin{bmatrix} \tilde{v}_{m_1} \\ \tilde{v}_{m_1+1} \\ \vdots \\ \tilde{v}_{m_2-1} \\ \tilde{v}_{m_2} \end{bmatrix}^{n+\frac{q}{L}} = \begin{bmatrix} \tilde{v}_{m_1} + c(\frac{q}{L} u_{m_1-1}^{n+1} + \frac{L-q}{L} u_{m_1-1}^{n}) \\ \tilde{v}_{m_1+1} \\ \vdots \\ \tilde{v}_{m_2-1} \\ \tilde{v}_{m_2} \end{bmatrix}^{n+\frac{q-1}{L}} . \tag{1.19}$$

1.2 Error Analysis

We consider here the situation represented by Fig. 1.4, and divide the local time stepping algorithms discussed so far into three categories to simplify the error analysis:

- Local time stepping in which the differences between solutions calculated using different time steps are ignored, and not all solutions are advanced to the same time level. We call this the traditional local time stepping method (TL method).

- Local time stepping in which the differences between solutions calculated using different time steps are ignored, but all solutions are advanced to the same time level using multiple sub-steps. We call this the modified local time stepping method (ML method).

- Local time stepping in which the differences between solutions calculated using different time steps are accounted for by using interpolations and extrapolations, and all solutions are advanced to the same time level using multiple sub-steps. We call this the new local time stepping method (NL method).

For the TL method, it is obvious from Fig. 1.2 that after n time steps, the solutions advanced using time step dt_1 will be defined at $T = n\mathrm{dt}_1$, while those advanced using time step dt_2 will be defined at $T = n\mathrm{dt}_2$. So the error will be of the order $O(n(\mathrm{dt}_1 - \mathrm{dt}_2))$, where n is the number of time steps and we have assumed $\mathrm{dt}_1 > \mathrm{dt}_2$.

For the ML method, all solutions are advanced to the same time level using multiple sub-steps in the temporal dimension. However, there is an error, in addition to the truncation errors in the original difference equation, caused by ignoring the differences between solutions calculated using different time steps. For example, the difference equation for the first substep at point m_1 is

$$\frac{\tilde{v}_{m_1}^{n+1} - v_{m_1}^{n}}{\mathrm{dt}_2} + \frac{\tilde{v}_{m_1}^{n+1} - \tilde{v}_{m_1-1}^{n+1}}{\mathrm{dx}_2} = 0. \tag{1.20}$$

If the difference of solutions at different time levels is not considered, then we actually have

$$\frac{\tilde{v}_{m_1}^{n+1} - v_{m_1}^{n}}{\mathrm{dt}_2} + \frac{\tilde{v}_{m_1}^{n+1} - u_{m_1-1}^{n+1}}{\mathrm{dx}_2} = 0, \tag{1.21}$$

which is what has been actually used in the traditional local time stepping algorithm. By using Taylor expansion, we can write Eq. (1.21) as

$$\frac{\tilde{v}_{m_1}^{n+1} - v_{m_1}^{n}}{\mathrm{dt}_2} + \frac{\tilde{v}_{m_1}^{n+1} - \tilde{v}_{m_1-1}^{n+1} + O(\mathrm{dt}_2)}{\mathrm{dx}_2} = 0, \tag{1.22}$$

or

$$\frac{\tilde{v}_{m_1}^{n+1} - v_{m_1}^{n}}{\mathrm{dt}_2} + \frac{\tilde{v}_{m_1}^{n+1} - \tilde{v}_{m_1-1}^{n+1}}{\mathrm{dx}_2} + \frac{O(\mathrm{dt}_2)}{\mathrm{dx}_2} = 0. \tag{1.23}$$

It is clear from Eq. (1.23) that the additional error term is $O(\frac{\mathrm{dt}_2}{\mathrm{dx}_2})$ at point m_1 for each sub-step. The situation for the grid point m_2+1 can be analyzed in a similar way, and leads to the same error estimate.

For the NL method, since we consider the difference between solutions at different time levels using a first order interpolation $\tilde{v}_{m_1-1}^{n+1} = \frac{1}{2}(u_{m_1-1}^{n+1} + u_{m_1-1}^{n})$, Eq. (1.20) can be written as

$$\frac{\tilde{v}_{m_1}^{n+1} - v_{m_1}^{n}}{\mathrm{dt}_2} + \frac{\tilde{v}_{m_1}^{n+1} - \frac{1}{2}(u_{m_1-1}^{n+1} + u_{m_1-1}^{n})}{\mathrm{dx}_2} = 0, \tag{1.24}$$

which can be re-written as

$$\frac{\tilde{v}_{m_1}^{n+1} - v_{m_1}^{n}}{\mathrm{dt}_2} + \frac{\tilde{v}_{m_1}^{n+1} - \tilde{v}_{m_1-1}^{n+1} + O(\mathrm{dt}_2^2)}{\mathrm{dx}_2} = 0. \tag{1.25}$$

The additional error is then $O(\frac{\mathrm{dt}_2^2}{\mathrm{dx}_2})$ at the grid point m_1. The situation at the grid point m_2+1 can be analyzed in a similar way.

From the above analysis, it appears that the local time stepping algorithms are no longer consistent, irrespective of whether the difference between solutions calculated using different time steps is considered or not. The additional truncation error at the grid point m_1 or m_2+1 is $O(\frac{\mathrm{dt}_2}{\mathrm{dx}_2})$ if we ignore the difference and $O(\frac{\mathrm{dt}_2^2}{\mathrm{dx}_2})$ if we consider the difference.

Gustafsson has shown [9] that for systems of hyperbolic equations, a stable numerical algorithm of order p will maintain its order of accuracy even if the boundary condition is approximated by a scheme of only order $p-1$. This result can be interpreted as the following: An error term of $O(\mathrm{dx}^{p-1})$ at the boundary generates an error that is one order higher, i. e. $O(\mathrm{dx}^p)$, inside the spatial domain. Based on this result, we can state the following:

- The additional error of the ML method at the grid point m_1 will have an effect of order $O(\frac{\mathrm{dt}_2}{\mathrm{dx}_2}\mathrm{dx}_2) = O(\mathrm{dt}_2)$ on the solution in the interior of the spatial domain.

- The additional error of the NL method at the grid point m_1 will have an effect of order $O(\frac{\mathrm{dt}_2^2}{\mathrm{dx}_2}\mathrm{dx}_2) = O(\mathrm{dt}_2^2)$ on the solution in the interior of the spatial domain.

The situation at the grid point m_2+1 is similar.

In general, if the time integration scheme is of order $O(\mathrm{dt}^p)$, the interpolation scheme needs to be at least of order $p-1$ in order to maintain the accuracy of the

algorithm used in the interior of the spatial domain, i. e.

$$\tilde{v}_{m_1-1}^{n+1} = \sum_{s=q}^{1} a_s u_{m_1-1}^{n+s} + O(\mathrm{dt}^p). \tag{1.26}$$

The two cases that we have discussed above correspond to

- 0-th order interpolation, $q = 1$, $a_1 = 1$, $\tilde{v}_{m_1-1}^{n+1} = u_{m_1-1}^{n+1} + O(\mathrm{dt})$,
- 1st-order interpolation, $q = 0$, $a_1 = a_0 = \frac{1}{2}$, $\tilde{v}_{m_1-1}^{n+1} = \frac{1}{2}(u_{m_1-1}^{n+1} + u_{m_1-1}^{n}) + O(\mathrm{dt}^2)$,

which are adequate for time integration algorithms of first and second order, respectively.

The analysis for the algorithm at point $m_2 + 1$ can be done similarly. If the time integration scheme is of order p, and $\mathrm{dt}_1 = L\mathrm{dt}_2$, then

- an extrapolation of order $p - 1$ is needed to approximate $u_{m_2}^{n+1}$ for the first substep, and
- an interpolation of order $p - 1$ is needed for the substeps 2 to $L - 1$ if numerical boundary conditions are required (for example with central difference schemes or system of equations having different characteristic directions).

1.3 Numerical Examples

We consider two numerical examples here to demonstrate the theoretical results obtained from the previous sections.

Table 1.1: Maximum errors using different methods at steady state

dx_1	0.1	0.05	0.025	0.01	0.005	0.0025	0.001
TL	9.54e-7	9.54e-7	9.54e-7	9.54e-7	9.54e-7	9.54e-7	9.54e-7
ML	2.38e-7	2.38e-7	2.38e-7	2.38e-7	2.38e-7	2.38e-7	2.38e-7
NL	1.48e-12	8.70e-13	4.66e-13	1.92e-13	9.91e-14	5.85e-14	4.66e-15

Table 1.2: Maximum errors using different methods ($T = 1.0$) for example 1.

dx_1	0.1	0.05	0.025	0.01	0.005	0.0025	0.001
TL	8.01e-2	8.56e-2	8.98e-2	9.28e-2	9.38e-2	9.43e-2	9.46e-2
ML	2.08e-2	1.27e-2	7.01e-3	3.02e-3	1.56e-3	7.94e-4	3.23e-4
NL	2.57e-2	1.52e-2	8.38e-3	3.61e-3	1.86e-3	9.48e-4	3.85e-4

Example 1:

$$u_t + u_x = 0, \qquad t > 0, \qquad x \in (0,1), \tag{1.27}$$

$$u(0,t) = -e^{-t}sin(t) + 1, \qquad t > 0,$$

$$u(x,0) = e^{x}sin(x) + 1, \qquad x \in [0,1].$$

The exact solution is $x = e^{(x-t)}sin(x-t) + 1$. The spatial grid is similar to that shown in Fig. 1.1, with dx_1 and dt_1 being used for [0.0, 0.4] and [0.6, 1.0], and dx_2 and dt_2 for [0.4, 0.6], respectively. Three different algorithms, corresponding to the three cases analyzed in the previous section, were used in the computation. The discretization scheme is backward in time and first order upwind in space.

The data in Table 1.1 shows that the errors are at the level of machine zero (single precision) for all three algorithms. There is no significant difference in

accuracy. This is because at $T = 25$, the solution has reached steady state. As discussed earlier in the paper, traditional local time stepping algorithms are adequate for calculating steady state solutions. The data in Table 1.2 shows, however, that the error from the TL method is significantly larger than that from the ML and the NL methods if a time T is chosen where the solution has not reached steady state. Since the original difference scheme used in the interior of the spatial domain is first order in time and space, the additional errors caused by local time stepping methods, which is $O(\mathrm{dt}_2)$ for the ML method and $O(\mathrm{dt}_2^2)$ for the NL method, are either at the same or higher order than the original truncation error. This explains why the ML method and NL method have similar accuracy in this example.

Example 2: The equation and the grid used in this example is similar to that in example 1:

$$u_t + u_x = 0, \qquad t > 0, \qquad x \in (0,1), \tag{1.28}$$

$$u(0,t) = -tanh(t), \qquad t > 0,$$

$$u(x,0) = tanh(x), \qquad x \in [0,1].$$

The exact solution is $x = tanh(x - t)$, which was plotted in Fig. 1.1. The discretization algorithm is second order accurate in both time and space using Crank-Nicolson and upwind methods.

The data in Table 1.3 shows that the errors from the TL method remains approximately constant. This is because the solutions at the grid points in $[0.4, 0.6]$

Table 1.3: Maximum errors using different methods ($T = 1.0$) for example 2.

dx_1	0.1	0.05	0.025	0.01	0.005	0.0025	0.001
TL	1.39e-1	1.55e-1	1.64e-1	1.69e-1	1.71e-1	1.72e-1	1.73e-1
ML	1.82e-2	9.45e-3	4.82e-3	1.95e-3	9.79e-4	4.91e-4	1.94e-4
NL	1.18e-3	3.26e-4	8.69e-5	1.47e-5	3.76e-6	9.54e-7	1.74e-7

were advanced using dt_2, which is only half of dt_1. While solutions at all other grid points were advanced to $T = 1.0$, these solutions were only advanced to $T = 0.5$!

It is also obvious from the data in Table 1.3 that the ML method, while being much more accurate than the TL method, is only first order accurate. As the grid sizes in both time and space were refined by two orders of magnitude, the error was also reduced by the same orders of magnitude. On the other hand, the NL method is clearly second order accurate in both time and space. The error is reduced by four orders of magnitude with the same grid refinement.

1.4 Non-Linear Equations

Till now, we have demonstrated that implicit local time stepping algorithms can be improved for calculating accurate *unsteady* solutions by using sub-steps and proper interpolations and/or extrapolations. Specifically, if the time integration algorithm is accurate to order p in the temporal dimension, then an interpolation (and/or extrapolation) of order $p - 1$ is needed to maintain accuracy.

Although it is difficult to implement this algorithm for practical computations with a large number of different time step sizes (or in the extreme case that the

time step size is different for each spatial grid point), it will be very efficient for problems with only a few significantly different spatial and temporal scales, such as resolving a sharp moving front in fluid dynamics calculations. The current work uses the accurate local time stepping algorithm for the simulation of detonation processes and the shock tube problem.

The analysis was simple for linear system of equations. There were not much issues regarding which states are to be used in interpolating and/or extrapolating. But this is not the case with a non-linear set of equations like the Navier-Stokes equations. Issues like whether to interpolate conservative variables or primitive variables arise for example, in the case of non-linear systems, the results may vary depending on which variables we are interpolating. This is because the behaviour of non-linear systems is not smooth. Peroomian and Charkravarthy [10] had shown that simple conservation variable limiting and interpolation would give rise to oscillations in pressures and eventually numerically negative pressures. So, in the present study, we have interpolated using the primitive variables.

The general algorithm is as follows. The domain of the flow is divided into two regions, the large time step region and the slow time step region. The flow is calculated with different time steps in the different time regions. This is the global time step. Now evidently, the temporal advancements in the two regions will be different because of the different time steps. In order to compensate for this difference, we do as explained before for sub time steps. For the sub time steps, we use the values calculated in the global time step and the values of the previous time step and interpolate and get better estimate of the primitive values. After

the solution in the sub time steps are obtained, these values are used along with the values obtained from the global time step to get the solution in the next time step. For the individual time steps (large time steps and the slow time steps),we perform individual Newton iterations and Gauss-Seidel sub iterations to obtain the solution in the different regions. This will be explained later in detail.

In the following, Chapter II introduces about the governing equations for a fluid flow with mixtures of gases in chemical non-equilibrium. The Numerical formulation is discussed in Chapter III. The Loci framework, in which the chemically reacting flow solver was developed, is discussed in Chapter IV. Chapter V discusses the results that were obtained using accurate local time stepping algorithm. Finally, conclusions and the scope of future work are discussed in Chapter VI.

CHAPTER II

GOVERNING EQUATIONS

This chapter discusses the equations that govern the flow of chemically reacting gases. The discussion that follows includes the transport terms, such as viscosity, heat conduction, and mass diffusion. However, details on the modeling of transport properties are omitted, because the cases investigated in this study are inviscid. Navier-Stokes equations are used to model the fluid motion. These equations are incomplete by themselves: they have to be completed by additional equations (closure equations) to produce a closed set of equations. The Navier-Stokes equations are derived using the continuum mechanics and the conservation laws. The derivation can be found in any standard text on Fluid mechanics.

2.1 Navier-Stokes Equations

The Navier-Stokes equations consist of three conservation laws: the laws of conservation of mass, conservation of momentum, and conservation of energy. The three laws can be put in the form of equations using the continuum field variables density, ρ, velocity, $\tilde{u}$, total energy, e_0, pressure, p, and temperature, T. The mass

conservation equation, in tensor form is:

$$\frac{\partial \rho}{\partial t} + div(\rho \tilde{u}) = 0, \tag{2.1}$$

The momentum conservation equation is given by:

$$\frac{\partial}{\partial t}(\rho \tilde{u}) + div(\rho \tilde{u}\tilde{u} + p\tilde{\tilde{I}}) = \rho \tilde{f} + div\ \tilde{\tilde{\sigma}}, \tag{2.2}$$

where the deviatoric stress tensor, $\tilde{\tilde{\sigma}}$, is given in Cartesian form as

$$\tilde{\tilde{\sigma}} = (\lambda div\ \tilde{u})\tilde{\tilde{I}} + \mu(grad\ \tilde{u} + (grad\ \tilde{u})^T). \tag{2.3}$$

In this equation, μ represents the fluid viscosity, while λ is normally derived from Stokes' Law which states that $3\lambda + 2\mu = 0$. The vector function, $\tilde{f}$ represents a body force such as a force exerted by gravitational or electro-magnetic means.

The final equation of the Navier-Stokes equations, the conservation of energy equation, is

$$\frac{\partial}{\partial t}(\rho e_0) + div((\rho e_0 + p)\tilde{u}) = div(\tilde{\tilde{\sigma}} \cdot \tilde{u}) + div(k\ grad\ T) + \rho\ (\tilde{f} \cdot \tilde{u}). \tag{2.4}$$

where $e_0 = \frac{1}{2}\tilde{u} \cdot \tilde{u} + e_{internal}$.

In this equation, k is the thermal conductivity of the fluid. The inviscid formulation of the Navier-Stokes equation is obtained by taking the limit of these

equations as μ approaches zero. Generally, k is also assumed to be zero for an inviscid formulation.

2.2 Species Continuity Equations

For each chemical species, the conservation of mass equation is

$$\frac{\partial \rho_s}{\partial t} + div(\rho_s \tilde{u}) = \dot{w}_s \tag{2.5}$$

where $\dot{w}_s$ is the rate of production or consumption of species s due to the chemical reaction(s). For this individual species equation to be consistent with the conservation of mass equation, the following result has to be true.

$$\sum_{s=1}^{NS} \dot{w}_s = 0 \tag{2.6}$$

2.3 Thermal Equation of State

As stated earlier, the Navier-Stokes equations are incomplete by themselves. They need a thermal and a caloric equation of state to form a closed set of equations. Assuming a mixture of thermally perfect gases, we can use Dalton's Law which states that the total pressure of a mixture of gases is the sum of the partial pressures,

$$p = \sum_{s=1}^{NS} p_s = \sum_{s=1}^{NS} \rho_s R_s T \tag{2.7}$$

where NS is the number of gas species in the mixture, R_s is the species gas constant, and ρ_s is the species density. The species gas constant is given by

$$R_s = \frac{\tilde{R}}{M_s} \tag{2.8}$$

where M_s is the species molecular mass, and $\tilde{R}$ is the universal gas constant.

2.4 Thermo-chemical models

The modeling of thermodynamic and chemical processes becomes a major concern when dealing with high temperatures. In most cases, when we have a mixture of gases, the gases behave as if they have no effect on each other at the molecular level. Thus, they can be considered as a mixture of thermally perfect gases. The chapter focusses on the thermodynamic and chemical models used in this work to describe fluid flows in chemical non-equilibrium.

2.4.1 Thermodynamic Models

Based on the fluid temperature and the molecular interactions within the fluid, four types of fluid flows can be identified [11]

Calorically Perfect Gas: by definition this gas has constant specific heats and thus a constant ratio of specific heats, $\gamma = \frac{C_p}{C_v}(\gamma = 1.4$ for air).

Thermally Perfect Gas: a thermally perfect gas is defined as one in which the specific heats are functions of temperature only. Consequently, γ varies with temperature only.

Mixture of thermally perfect gases: components are thermally perfect, and the mixture is thermally perfect if no chemical reactions are present. The mixture is *not* thermally perfect in the presence of chemical reactions and the enthalpy and energy of the mixture are a function of temperature and flow composition.

Real Gas: In this case, intermolecular forces must be included. A real gas occurs in the presence of very high pressures or low temperatures. The mixture enthalpy and energy are now functions of temperature and a second state variable (e.g. density or pressure).

A real gas is usually not considered to be chemically reacting. It should be noted that sometimes in the scientific literature the term *real gas* has been used to denote mixtures of chemically reacting, thermally perfect gases, in most cases with the added assumption of local chemical equilibrium[12]. In this study, the fluid will be considered to be a mixture of thermally perfect gases with chemical non-equilibrium (e.g. finite rate chemistry).

2.4.2 Caloric Equation of State

The thermal equation of state (2.7) relates pressure to temperature. The governing equations introduced earlier are expressed in terms of conserved variables, such as momentum and energy, and do not directly describe temperature, which is determined from the internal energy of the gas. The governing equations remain valid for flows where high temperatures exist, but it becomes necessary to modify the caloric equation of state from the simple, calorically perfect, low

temperature model. In many cases, high temperatures cause the onset of chemical reactions, which lead to the dissociation of molecules and ionization of neutral species. For a gas consisting of a mixture of species, the caloric equation of state is computed by writing the mixture internal energy, $e_{internal}$, as the sum of the species energies, as in

$$e_{internal} = \sum_{s=1}^{NS} Y_s e_s. \tag{2.9}$$

For ideal gases, the relationship between internal energy and temperature is linear. However, for reacting gases, rotational and vibrational modes of the molecules tend to bring up a non-linear relation between the internal energy and the temperature. The general form of the equation for species energy is given by,

$$e_s = \int_{T_{ref}}^{T} c_{v_s}(\tau) d\tau + h_{f_s}, \tag{2.10}$$

where h_{f_s} is the species heat of formation, or the energy required to create that species at T_{ref}, and c_{v_s} is the specific heat at constant volume for species s. If the translational, rotational and vibrational components are assumed to be in thermodynamic equilibrium, and the vibrational mode is modeled using a simple harmonic oscillator[13], then the species energy is given as,

$$e_s = n_s R_s T + \sum_{v=1}^{NVT_s} \frac{R_s \theta_{v,s}}{e^{\theta_{v,s}/T} - 1} + h_{f_s}, \tag{2.11}$$

where NVT_s is the number of the vibrational modes, $\theta_{v,s}$ is the vibrational temperature(s), and n_s specifies the translational and rotational contributions to internal energy.

2.4.3 Finite rate chemistry

At this point, the inviscid governing equations form a set of closure equations when the species production rates,$\dot{w}_s$, are specified for a general chemistry model. The generic equation for a chemical reaction (assuming that NR reactions are occurring simultaneously) is given by

$$\sum_{m=1}^{NS} \nu'_{m,r} X_m \rightleftarrows \sum_{m=1}^{NS} \nu''_{m,r} X_m \quad for\ r = 1, 2, \ldots, NR \tag{2.12}$$

Where X_m is a chemical species in the fluid, $\nu'_{m,r}$ are the stoichiometric coefficients for the reactants and $\nu''_{m,r}$ are the ones for the products (both related to species m in reaction r). From stoichiometry, the rate of change for a given species, i, can be defined as [14]

$$\dot{w}_s = \left(\frac{d\rho_s}{dt}\right)_{chemistry} = M_s \sum_{r=1}^{NR} (\nu''_{s,r} - \nu'_{s,r}) \left[k_{f,r} \prod_{l=1}^{NS} \left(\frac{\rho_l}{M_l}\right)^{\nu'_{l,r}} - k_{b,r} \prod_{l=1}^{NS} \left(\frac{\rho_l}{M_l}\right)^{\nu''_{l,r}} \right] \tag{2.13}$$

where $k_{f,r}$ is the forward reaction rate for reaction r, $k_{b,r}$ is the backward reaction rate, and NS is the number of species in the fluid. The forward reaction rate is usually described, for a given reaction r, by an Arrhenius-like equation,

$$k_{f,r} = C_{f,r} T^{\eta_{f,r}} e^{-\theta_{f,r}/T} \tag{2.14}$$

The coefficients C_f, η_f *and* θ_f have been obtained experimentally for many different reactions over a wide range of temperatures. We have a similar equation for the backward reaction rate, $k_{b,r}$. Both rates depend on accurate experimental data in order to provide valid results. The equilibrium rate constant, $k_{c,r}$, can be used in lieu of the backward rate reaction, and is related to the two previous quantities as follows

$$k_{c,r} = \frac{k_{f,r}}{k_{b,r}}.$$

Using the above equations, equation (2.13) can be rewritten as:

$$\dot{w}_s = M_s \sum_{r=1}^{NR} (\nu''_{s,r} - \nu'_{s,r}) k_{f,r} \left[\prod_{l=1}^{NS} \left(\frac{\rho_l}{M_l} \right)^{\nu'_{l,r}} - \frac{1}{k_{c,r}} \prod_{l=1}^{NS} \left(\frac{\rho_l}{M_l} \right)^{\nu''_{l,r}} \right] \tag{2.15}$$

The equilibrium rate constant, like the forward and backward rates, can be represented in an Arrhenius form using experimental data. We can also find the equilibrium rate of a reaction based on the minimization of the Gibbs free energy at constant pressure and temperature. This method is used in this study. The advantage of using this method is that we need not depend on the experimental results but we just need to get accurate reference thermodynamic values. The bad side is that this method is sensitve to the thermodynamic model that we use and will be affected by any of its shortcomings. The expression for the thermodynamic $K_{c,r}$ presented here are borrowed from the work of Carey Cox[12].

CHAPTER III

NUMERICAL FORMULATION

3.1 Spatial Discretization

The model equations expressed in integral form for an inviscid gas are :

$$\int_{\Omega(t)} \frac{\partial}{\partial t}\rho_s \, dV + \int_{\Omega(t)} div(\rho_s \tilde{u}) \, dV = \int_{\Omega(t)} \dot{w}_s \, dV,$$

$$\int_{\Omega(t)} \frac{\partial}{\partial t}(\rho \tilde{u}) \, dV + \int_{\Omega(t)} div(\rho \tilde{u}\tilde{u} + p\tilde{\tilde{I}}) \; dV = 0, \tag{3.1}$$

$$\int_{\Omega(t)} \frac{\partial}{\partial t}(\rho e_0) \, dV + \int_{\Omega(t)} div((\rho e_0 + p)\tilde{u}) \, dV = 0.$$

An identity for taking the time derivatives of integrals over general moving and deforming domains [15],

$$\frac{d}{dt}\int_{\Omega(t)} f(\tilde{x}, t) dV = \int_{\Omega(t)} \frac{\partial f}{\partial t} \; dV + \int_{\Omega(t)} f \tilde{u}_{\Omega}.\tilde{n} \; dS \tag{3.2}$$

is useful. Using Green's theorm and the identity 3.2, the above equations can be transformed into the following set of equations :

$$\frac{d}{dt}\int_{\Omega_c(t)} Q \; dV + \int_{\partial\Omega_c(t)} F \; dS = \int_{\Omega_c(t)} \dot{W} \; dV, \tag{3.3}$$

where the conservative state variables, Q, inviscid flux function, F, and chemistry source term, $\dot{W}$, are given by

$$Q = \begin{bmatrix} \rho_1 \\ \vdots \\ \rho_s \\ \vdots \\ \rho_{NS} \\ \rho\tilde{u} \\ \rho e_0 \end{bmatrix}, F = \begin{bmatrix} \rho_1(\tilde{u} - \tilde{u_\Omega}).\tilde{n} \\ \vdots \\ \rho_s(\tilde{u} - \tilde{u_\Omega}).\tilde{n} \\ \vdots \\ \rho_{NS}(\tilde{u} - \tilde{u_\Omega}).\tilde{n} \\ (\rho\tilde{u}(\tilde{u} - \tilde{u_\Omega}) + p\tilde{\tilde{I}}).\tilde{n} \\ (\rho e_0(\tilde{u} - \tilde{u_\Omega}) + p\tilde{u}).\tilde{n} \end{bmatrix}, \dot{W} = \begin{bmatrix} \dot{w}_1 \\ \vdots \\ \dot{w}_s \\ \vdots \\ \dot{w}_{NS} \\ 0 \\ 0 \end{bmatrix}$$

and Ω represents the general control volume, $\partial\Omega$ is its surface, $\tilde{u}_\Omega$ is the surface velocity, and $\tilde{n}$ is the outward pointing surface normal. The subscript c, refers to a generic cell.

Equation (3.3) is integrated by approximating the volume and the surface integrals. Mid-point rule is used for the volume integrals, and for the surface integrals, the contributions from each of the NF faces of the cell c is added.

Assuming that $Q_c(t)$ is the value of Q at the centroid of the cell C, and $\nu_c(t)$, defined by,

$$\nu_c(t) = \int_{\Omega_c(t)} dV, \tag{3.4}$$

the numerical integration of Q using the mid-point rule becomes,

$$\int_{\Omega_c(t)} Q(\tilde{x}, t)dV = Q_c(t)\nu_c(t). \tag{3.5}$$

For the numerical computation of the surface integral of equation (3.3), assume that the flux can be approximated by a function, $\hat{F}(Q_l, Q_r)$, of conservative values from the left and right of the face. From this,

$$\int_{\partial\Omega_c(t)} F dS = \sum_{f=1}^{NF_c} \int_{\partial\Omega_{c,f}(t)} F dS \approx \sum_{f=1}^{NF_c} A_{c,f}(t)\hat{F}(Q_{l,f}, Q_{r,f}), \tag{3.6}$$

where the area of the face, $A_{c,f}(t)$, is defined as

$$A_{c,f}(t) = \int_{\partial\Omega_{c,f}(t)} dS. \tag{3.7}$$

Using the previous equations, the equation (3.3) can be rewritten as,

$$\frac{d}{dt}[\nu_c(t)Q_c(t)] + \sum_{f=1}^{NF_c} A_{c,f}(t)\hat{F}(Q_{l,f}, Q_{r,f}) = \nu_c(t)\dot{W}_c(t). \tag{3.8}$$

Applying chain rule for the first term in the equation (3.8),

$$\frac{d}{dt}[\nu_c(t)Q_c(t)] = Q_c(t)\frac{d}{dt}\nu_c(t) + \nu_c(t)\frac{d}{dt}Q_c(t). \tag{3.9}$$

The derivative of volume with respect to time can be converted into a spatial integral through the use of the identity, (3.2) as,

$$Q_c\frac{d}{dt}\nu_c(t) = Q_c\frac{d}{dt}\int_{\Omega_c(t)} dV = Q_c\int_{\partial\Omega_c(t)} \tilde{u}_\Omega.\tilde{n}dS \approx Q_c\sum_{f=1}^{NF_c} A_{c,f}(t)(\tilde{u}_{\Omega,f}.\tilde{n}_{c,f}).$$

This term is important in case of deforming mesh (in the case of non-deforming mesh, this quantity is zero). Using the information presented, the governing

equations can be written as a set of ordinary differential equations of the form,

$$\frac{d}{dt}Q_c = R_c, \tag{3.10}$$

where R_c is given by,

$$R_c = \frac{1}{\nu_c}[\nu_c \dot{W}_c - \sum_{f=1}^{NF_c} A_{c,f}\hat{F}(Q_{l,f}, Q_{r,f}) - Q_c \sum_{f=1}^{NF_c} A_{c,f}(\tilde{u}_{\Omega,f}.\tilde{n}_{c,f})]. \tag{3.11}$$

3.2 Temporal Discretization

Numerical integration of the system of equations, (3.10) and (3.11) is performed using the family of schemes introduced in [16]. These schemes are described by the time-discretized equation

$$\frac{(1+\psi)\Delta Q^n - \psi \Delta Q^{n-1}}{\Delta t} = (1-\theta)R^n(Q^n) + \theta R^{n+1}(Q^{n+1}), \tag{3.12}$$

where

$$\Delta Q^n = Q^{n+1} - Q^n.$$

In the above equation, n represents the time level and θ, ψ are parameters that control the accuracy of the discretization. Equation (3.12) represents a set of implicit time integration schemes, including the second-order three point backward ($\theta = 1, \psi = \frac{1}{2}$), backward Euler ($\theta = 1, \psi = 0$), and Crank-Nicholson ($\theta = \frac{1}{2}, \psi = 0$). It also includes explicit schemes like the forward Euler ($\theta = 0, \psi = 0$).

The solution of (3.12) for Q^{n+1} given Q^n is the goal of the time integration procedure. However, the solution is complicated by the fact that $R^{n+1}(Q^{n+1})$ is a non-linear function with a non-trivial inverse. So, instead of solving for (3.12) directly, the common approach is to use Newton iterative method for the solution of the non-linear homogenous equation given by

$$L(Q^{n+1}) = Q^{n+1} - Q^n - \frac{\Delta t}{1+\psi}[(1-\theta)R^n(Q^n) + \theta R^{n+1}(Q^{n+1})] - \frac{\psi}{1+\psi}(Q^n - Q^{n-1}) = 0. \tag{3.13}$$

The vector form of the Newton method is used to obtain the zero for the vector valued function, $L(Q)$. Consequently, the Newton method should converge to the vector value of Q^{n+1}. The Newton method proceed by iteratively solving the equation

$$L'(Q^{n+1,p})(Q^{n+1,p+1} - Q^{n+1,p}) = -L(Q^{n+1,p}),\ p \geq 0, \tag{3.14}$$

where the Newton iteration is initialized using the previous time-step values, thus $Q^{n+1,p=0} = Q^n$. Assuming that volumes, areas, and surface velocities are not functions of Q, then the Jacobian, $L'(Q^{n+1,p})$, is given by,

$$L'(Q^{n+1,p}) = I - \frac{\theta \Delta t}{1+\psi}\left[\frac{\partial}{\partial Q} R^{n+1}(Q)\right],$$

or,

$$L'(Q^{n+1,p}) = I - \frac{\theta \Delta t}{1+\psi}\left[\frac{\partial \dot{W}}{\partial Q} - \sum_{f \in faces} \frac{A_f^{n+1}}{\nu^{n+1}} \frac{\partial \hat{F}(Q_{l,f}, Q_{r,f})}{\partial Q} - I \sum_{f \in faces} \frac{A_f^{n+1}}{\nu^{n+1}} (\tilde{u}_{\Omega,f} \cdot \tilde{n}_f)\right]. \tag{3.15}$$

The Jacobian in equation (3.15) is a sparse matrix with dense sub-blocks. The off-diagonal blocks are terms generated by the flux Jacobians, due to their functional dependence on their neighboring cells.

3.3 Linear System Solution

Each Newton iteration step, given by equation(3.14), requires solving a linear system of equations of the form

$$Ax = b,$$

where

$$A = L'(Q^{n+1,p}),\ x = (Q^{n+1,p+1} - Q^{n+1,p}),\ b = -L(Q^{n+1,p}).$$

As already mentioned, the matrix, A of this linear system is given by equation (3.15), and is typically a matrix with sparse structure that is composed of dense sub-blocks. Most of the terms of equation (3.15) contribute to the diagonal block of the matrix, while the flux terms, being a function of left and right values of Q, contribute to the off-diagonal terms. Inviscid flux Jacobian terms are treated as Jacobians of the first order functions in an effort to increase the sparseness of the matrix A, and similar steps are taken for the viscous flux Jacobians. The matrix A can be factored into lower, upper and diagonal blocks, as in

$$A = L + D + U$$

.

For first order flux Jacobians, each internal face of the mesh contributes a term to the diagonal of the cells on either side, while also contributing one block to the lower matrix, L and one block to the upper matrix, U. The system of equations is solved using a symmetric Gauss-Seidel method. Iterative methods of this form are rather efficient at solving systems of equations produced by finite volume schemes applied to hyperbolic equations, such as the fluid dynamics equations presented here. The symmetric Gauss-Seidel iterative solver works by a two-pass method. These two passes, a forward and a backward pass, are a consequence of the solution of the factored equations,

$$(L + D)x^{*i+1} + Ux^i = b, \ and$$

$$Lx^{*i+1} + (D + U)x^{i+1} = b, \tag{3.16}$$

where x^{*i+1} is the result of the forward pass of the symmetric Gauss-Seidel iteration. The iteration is initialized with the first pass of a block Jacobi iterative method given by

$$x^0 = D^{-1}b. \tag{3.17}$$

In the solution of equations (3.16) and (3.17), a dense GAXPY(general A x plus y) LU method is employed to invert the diagonal blocks of D. This LU factorization is performed once before the Gauss-Seidel iteration proceeds and is used in both passes.

3.4 Flux Computation and the Jacobian Matrix

The residual equation (3.11) is obtained by evaluating the spatial terms. The present discussion neglects the geometric conservation term. The source term variable, *src* defined by

$$src = \nu \dot{W} - \sum_{f=1}^{NF} A_f \hat{F}(Q_{l,f}, Q_{r,f}). \tag{3.18}$$

identifies the spatial integrals. The first term consists of the cell volumes and the species production rates (which can be obtained from equation (2.15)). The second term can be obtained by summing up the fluxes about the faces of a cell.

The Jacobian described by equation (3.15), consists of a diagonal block matrix combined with off-diagonal matrices which are formed from the differentiation of the flux function. With the flux function, $\hat{F}(Q_{left}, Q_{right})$, defined by the conservative variables on the left and right side of the face, the Jacobian of this function can be given by,

$$fjp = A^{n+1} \frac{\partial \hat{F}(Q_{left}, Q_{right})}{\partial Q_{left}},$$

$$fjm = A^{n+1} \frac{\partial \hat{F}(Q_{left}, Q_{right})}{\partial Q_{right}}.$$

The variables, **fjp** and **fjm**, are Jacobians of the flux functions located at faces and are components of the overall Jacobian matrix given in equation (3.15). The

diagonal blocks of the Jacobian can be obtained from the following relation:

$$D_c = I - \frac{\theta \triangle t}{\nu^{n+1}(1+\psi)} \left[\nu^{n+1} \frac{\partial \dot{W}_c}{\partial Q_c} - \sum_{f \rightarrow left=c} (fjp)_f + \sum_{f \rightarrow right=c} (fjm)_f \right] \quad (3.19)$$

Equation (3.19) gives the diagonal of the Jacobian, the **fjp** and **fjm** form the off-diagonal terms. A face contributes **fjp** to the left face diagonal and also contributes **fjp** to the right face's row equation. Similar effects are made by the **fjm** terms. The orientation of the faces can be chosen in such a way that the Jacobian matrix always has **fjm** terms in its upper diagonal and **fjp** terms in its lower diagonal. Once the lower and the upper portions of the matrix are identified, a row based data structure for the matrix, based on the face connectivities can be formed. A detailed description of this data structure is given in [17].

3.5 Gauss-Seidel Iteration

The Gauss-Seidel algorithm is described by the equations (3.16). Recursive rule specification is used in evaluating the solution of the linear system. As discussed before, the algorithm consists of the forward pass and the backward pass. The present discussion focusses on the forward pass which evaluates the following equation:

$$x^{k+1} = -(L+D)^{-1}(Ux^k - b), \quad (3.20)$$

where $b = -L(Q^{n+1,p})$, and L, D, and U are the lower, diagonal and the upper components of the Jacobian matrix, respectively. This method can be written as

the point-wise recurrence relation,

$$x_i^{k+1} = D_i^{-1}\left(b_i - \sum_{j=1}^{i-1} L_{ij}x_j^{k+1} - \sum_{j=i+1}^{n} U_{ij}x_j^k\right).$$

Using the data structures described earlier, equation (3.20) can be rewritten as

$$x_i^{k+1} = D_i^{-1}\left[b_i + \frac{\theta\triangle t}{\nu^{n+1}(1+\psi)}\left(\sum_{(i,j)\in lower}(fjp)_j x_{cl_j}^{k+1} - \sum_{(i,j)\in upper}(fjm)_j x_{cr_j}^{k}\right)\right].$$

3.6 Accurate Local Time Stepping

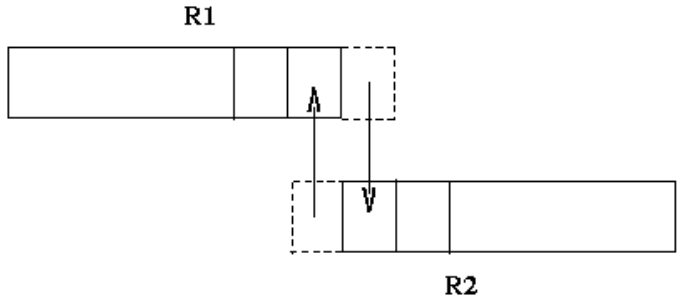

Figure 3.1: Mapping of the clone cells

The domain is divided into two regions, the fast region and the slow region. In the fast region, the time step(δt) is larger than that of the slow region; consequently the solutions obtained in these two regions are different in the temporal dimension. To explain the algorithm, it is assumed that there are only two regions as shown the figure 3.1. Region R1 is the fast time step region and region R2 is the slow time step region. The dotted cells represent the clone cells. The arrow indicates that they are actually mapped to the corresponding cells in the different regions and they are actually nothing but the cells they are mapped into. After the first

global time step, R1 will advance further in time than in region R2. So, we do a sub time step for just the region R2. In order to get the new L, D, U, b, we use the information from the previous time step (and if this is the first step, N=0, we use the initial conditions itself) and the information obtained from the global time step. Thus, after getting the new L, D, U, b, for the sub time steps, we obtain the solution for the slow time region. After the solution in the slow time region has advanced enough, i.e., corresponding to the global time step, the solution is advanced to the next time level. For the sub time steps, the initial conditions are taken from the resulting global time step for the slow time region. The other difference between the global time step and the sub time steps was the treatment at the boundaries. For the sub time steps, the boundary conditions were obtained from the clone cells. The process of interpolation, to obtain the boundary conditions can be better explained using the figure (3.2). The time stamps indicate

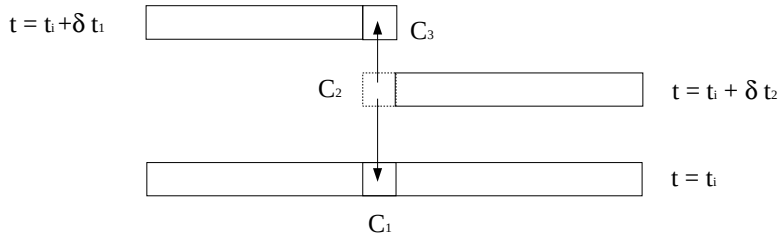

Figure 3.2: Interpolation of the primitive variables

the different temporal levels, each block will be in before and after the global time step. The time level, $t = t_i$ corresponds to the time level before the global time step while the time level, $t = t_i + \delta t_1$ corresponds to the time level of the fast time region after the global time step, and the time level $t = t_i + \delta t_2$ corresponds to the time level of the slow time region after the global time step. The cells C_1,C_2,

and C_3 represent the same cell in their corresponding temporal dimension. The arrows indicate that in order to estimate the primitive values at the cell C_2 the primitive values at cells C_1 and C_3 are interpolated. After interpolating, the second block was treated as a separate domain for which the boundary conditions were derived from the clone cells. A new set of flux computations were done and a new Jacobian matrix, corresponding to the slow time region was obtained. A solution vector corresponding to the new sub time level was obtained with the new L, D, U and b matrices. This process is repeated till all the cells in the slow time region were advanced (in temporal dimension) to the same extent as that of the cells in the fast time region ($t = t_i + \delta t_1$).

CHAPTER IV

LOCI

This chapter discusses the *Loci* framework, which was used to develop the code for the study, *Chem*. The *Chem* code is a chemically reacting compressible flow solver for simulating high speed and combustion flow problems. As a part of the present work, accurate local time stepping was implemented in the solver, using the Loci framework. The material covered in this chapter is a summary of the more detailed discussion in [17, 18].

4.1 Introduction to the Loci Framework

The Loci system is an application framework that seeks to reduce the complexity of assembling large-scale finite-difference or finite-element applications, although it could be applied to many algorithms that are described with respect to a connectivity network or graph. The complexity involved during the development of a large-scale computational field simulation can be attributed to the control and data movement. The Loci system focusses on this aspect and thus minimises the number of bugs in the system.

The gradual evolution of an application over time gives rise to inconsistencies between various application components that are developed during the life time of

a project. The inconsistencies can be attributed to the various persons involved in the development. All these problems are addressed by the Loci framework. The framework automatically generates the data and control movement operations of an application from component specifications, while guaranteeing a level of consistency between the components.

4.2 Data Models

The most fundamental concept in the Loci system is the entity. Computational graphs are represented by collections of entities and collections of maps or connectivity lists. Entities represents the sites where computations may occur in the graph. For example, the entities in a finite-volume calculation may represent faces, cells, and nodes of the mesh. Maps may connect faces to their left and right cells, or cells to their nodes, and so forth. Values are bound to the entities via the **store** construct, which provides an association from entities to values. The **parameter** construct provides a singleton interface to value, where a set of entities is associated to a single value. Relationships between entities are provided by the **map** construct. The map construct can be composed with the store construct to provide an abstraction of indirection. The **constraint** construct, used to constrain computations to some subset of entities, provides an identity mapping over a given set of entities. These constructs are illustrated in figure 4.1.

The database of facts that describe the problem are formulated by using these basic constructs. The database of facts is the starting point for logic programming systems, in the sense that the facts that are already there (obtained from the

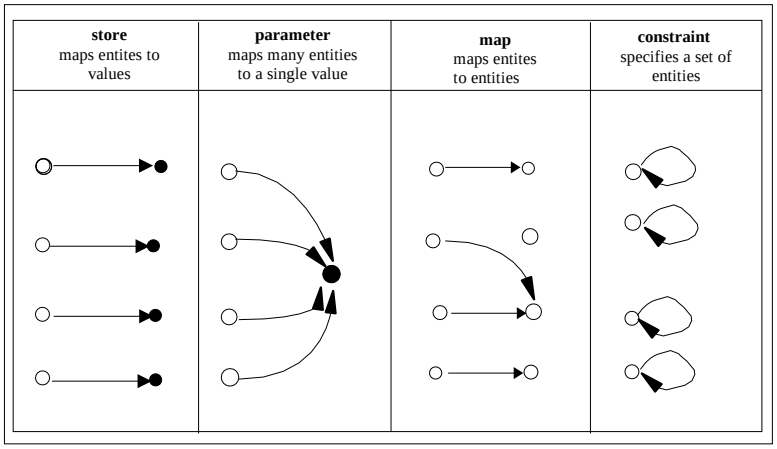

Figure 4.1: Four Basic Database Constructs

definition of the problem) are used in creating new facts (which are in turn the results of the rule applications) with the help of what is called as the *rule database* (described later). Thus the database is a center of communication for programs derived from the specifications. It should be noted that, although the term database might be associated with files stored on disk, here it refers to a model of data and associated data structures and the methods (*rules*) that are used to manipulate these data structures. Each fact provides information about some subset of entities, such as positions of nodes, or maps relating cells to nodes. Each of these facts is given an identifier that consists of a name, an iteration label, and an iteration offset. The iteration offsets are used to access the information from a previous iteration. The iteration label corresponds to the nested iteration levels of a loop. They are of the general form of $\alpha\{\tau + \theta\}$ where α is the name, τ is the iteration identifier, θ is the iteration offset. For example, $pressure\{n-1\}$ is the pressure at a previous iteration of iterator n. The "$\rightarrow$" operator is used to

represent the application of a map in the access of values. For example, a map represented as **value[left[f]]** in C language, is represented as **left→value**.

4.3 Rule Specifications

As described earlier, the database consists of two parts: the database of facts that includes the problem specification, and the database of rules that describes the transformations that can be used to introduce/compute new facts into the database. These rules correspond to fundamental computations involved in solution algorithms, such as rules for evaluating areas of faces, or for solving equations of state. These rules are specified using text strings, called rule signatures. These signatures describe the four basic database constructs described earlier. Rule signatures are of the form **head←body**, where head consists of a list of variables that are generated by the application of the rule, while the body contains a list of variables that are accessed while performing the computation. For example, $p \leftarrow \rho, T, R$ represents that a value for pressure p, is provided when values for density, ρ, temperature, T, and gas constant, R are present. The rule signature may also contain a mapping operator,"$\rightarrow$". For example, the rule $area \leftarrow face2nodes \rightarrow position$ indicates that areas are generated using the position of the nodes making the face and that "**face2nodes**" is a mapping connecting the faces to the corresponding nodes. The following sections discuss the various classes of computations that are encountered in a typical unstructured grid computation.

4.3.1 Rule Constraints

The rule constraints see to it that a particular rule can be applied to a certain subset of entities. In addition to this, it also asserts that the rules be applied to all the entities satisfying the constraints. In many cases this can be used to automatically detect inconsistencies caused by incomplete information. For example, an improperly specified boundary condition yields a rule that cannot satisfy its constraint, due to insufficient information. In addition, Loci identifies over-specification by requiring that each entity has only one value. Thus, if a boundary condition is applied to an interior node of the domain, then an error would result, caused by a conflict between the boundary condition and stencil specifications.

4.3.2 Point-wise rules

The most common rule involved in the numerical solution of PDE is the point-wise rule. The point-wise rule represents an entity-by-entity computation of values that are placed in the stores listed in its head. The semantics of the point-wise rule application requires that an output variable can only define one value per entity. If an ambiguous specification produces two rules that compute values for the same entity, an error is indicated during scheduling. Recursion is allowed in point-wise rules, provided that the *"single value per entity"* criterion is not violated. Thus, recursion in point-wise rules is bounded by the number of entities in the simulation mesh.

4.3.3 Reduction Rules

A reduction is described by a function composed of three components: a function that is applied to a set of values, an associative and commutative operator $\oplus$ that is defined on the type returned by the above mentioned function, and an identity element of operator $\oplus$, e. Thus a reduction, r, over values, $\{v_i | i \in [1, N]\}$, using function f and operator $\oplus$ is defined as

$$r = f(v_1) \oplus f(v_2) \oplus \cdots \oplus f(v_i) \oplus \cdots \oplus f(v_N)$$

When the reduction is evaluated using a left or right precedence rule, then a sequential evaluation is derived; however, the associative property of $\oplus$ allows for different parallel evaluation orders. For example, the set of values can be partitioned into subsets that can be evaluated concurrently as in

$$r = \{e \oplus f(v_1) \oplus f(v_2) \oplus \cdots \oplus f(v_p)\} \oplus \{e \oplus f(v_{p+1}) \oplus f(v_{p+2}) \oplus \cdots \oplus f(v_N)\}.$$

Parallel partitioning of reduction operations can be expressed when given three basic computational methods, identified as *unit, apply, and join*, as listed in table 4.1. The unit rule initializes a reduction variable to the identity element, the apply rule "accumulates" the partial results. The algorithm for partitioning this reduction operation among parallel processors is accomplished by creating a copy of the reduction variable on each processor participating in the reduction. Each reduction variable is initialized to the identity, e, and then followed by

Table 4.1: Reduction Specification in Loci

Rule Type	Function	Rule Signature
Unit Rule	$r_i^0 = e$	$r \leftarrow CONSTRAINT(v), UNIT(e)$
Apply Rule	$r_i^{j+1} = r_i^j \oplus f(v_i)$	$r \leftarrow r, v, APPLY(\oplus)$
Join Op	$r_i^{m+n} = r_i^m \oplus r_i^n$	Derived (No signature)

the application of all apply rules that are in that processor's partition. Finally, the partial results for each processor are reduced to the final result using join operations.

4.3.4 Iteration Rules

Iteration is defined by way of three types of rule specifications, the build rule, the advance rule and the collapse rule.

- *build rules* that construct the iteration,
- *advance rules* that advance the iteration,
- *collapse rules* that terminate the iteration.

This specification follows an analogy to the inductive proof: build rules are analogous to an inductive base, while advance rules are analogous to an inductive hypothesis. For example, an iteration where a variable named q is iterated to a converged solution may be described by the following three rules: 1) a build rule of the form $q\{n = 0\} \leftarrow ic$, 2) an advance rule similar to $q\{n + 1\} \leftarrow q\{n\}, dq\{n\}$, and 3) an iteration collapse rule $solution \leftarrow q\{n\}, CONDITION(converged\{n\})$. Iteration in this example proceeds by initializing the first iteration, $q\{n = 0\}$,

using the build rule (initial conditions). Next, termination of iteration begins with checking for the convergence: if the test succeeds then the collapse rule terminates the iteration. Finally, the iteration advances in time by the repeated application of the advance rule. Note that the completion of these rules may require invoking other rules specified in the rule database. In this case, rules that compute $converged\{n\}$ and $dq\{n\}$ will also need to be scheduled. To support iteration, variables that exist in lower levels of the iteration hierarchy are automatically promoted up the iteration hierarchy. Thus a variable that is computed in $iteration\{n\}$ is communicated to $iteration\{n, it\}$ automatically. In addition, rules that are specified completely at the stationary level will be promoted to any level of the hierarchy. This allows for the specification of relations that are iteration independent (for example, $p = \rho\tilde{R}T$ implies $p^n = \rho^n\tilde{R}^nT^n$ for an iteration at time level n).

4.4 Scheduling

In Loci, the mesh, boundary conditions, initial conditions, and other modeling information are stored in a database of facts using stores, parameters, maps and constraints. Scheduling occurs in three steps. The first step involves creating a dependency graph that connects the variables stored in the fact database to the goal, using the rules in the rule database. This step involves iteratively exploring the space of known variables and determining which rules can apply, which may in turn generate new variables. Once this graph is produced, it is pruned to only those rules that generate the requested goal, sorted into iteration hierarchies,

and reduced to a directed acyclic graph (DAG) by clumping recursive dependency loops. The next step is an existential deduction phase, which determines what attributes can be assigned to which entities. For example, the rule $p \leftarrow \rho, R, T$ specifies that entities that have attributes ρ, R, and T also have the attribute p. The existential deduction begins with the given facts, and follows the DAG topological order, computing the entities associated with each attribute until the goal is reached. During this existential deduction, recursive loops are iteratively evaluated until all possible attributes are generated. The result of the existential deduction phase is a concurrent schedule that obtains the requested goal. However, since it is possible that some attributes may exist for entities that don't contribute to the requested goal, a final optimization step prunes this schedule. The pruning operation starts from the goal and works backwards through the DAG, until the schedule only computes those values that are needed to provide the requested goal. A concurrent schedule is automatically produced from the scheduling process. Only partitioning of entities to processors is needed to generate a schedule for parallel processors (on distributed memory architectures, a communication schedule would also need to be deduced). Thus, the numerical model does not have any references to parallel execution – this arises naturally from the specification.

4.5 Implementation

The Loci system uses shallow inheritance hierarchies combined with templated containers and composers in its design strategy. The most basic data type for the Loci system is the *entitySet*, a value class that describes arbitrary sets of

entities. Fast intersection, union, and complement operations are provided for these entity sets. These entity sets are necessary for the existential deduction phase of scheduling, and used for controlling and allocating functions. The data models described in section (4.2) are implemented as templated container classes. These containers provide features that facilitate their storage in the fact database and their automatic binding in rule invocations. The rules described in section (4.3) are implemented via a shallow inheritance hierarchy. Users create new rules by providing a constructor, which creates the rule specification, and a virtual member function called **compute**, which provides a collection level interface to rule evaluations. The scheduler generates a partial order of entity computations for each rule invocation represented by the **sequence** class. An inlined member function is provided which performs the specific computations required by a single entity. A templated composer function, **do_loop**, provides the interface between the per-entity computation provided by the user and the collection-level request made by the execution schedule (in the form of an instance of **sequence**). This step optimizes in the sense that while the scheduler can concern itself with global analysis, the **do_loop** composer can provide the optimal looping interface. Most importantly, all of these optimizations can occur without change to the numerical algorithm as specified in the rule definition. For a detailed understanding of these topics, please refer to [17, 18].

CHAPTER V

RESULTS

This chapter discusses the results obtained from the test cases, which compared the accurate local time stepping results with those obtained from the original *Chem* code. The two problems dealt with were the shock tube and the detonation process.

5.1 The Shock Tube Problem

The shock tube was chosen because of the presence of several physical phenomenoa such as shocks, expansions, and contact surfaces. It is also suitable for evaluating simulations of the unsteady propagations of discontinuities. The shock tube flow can be obtained by the sudden rupture of the diaphram in a long tube separating two regions, filled with gases at different initial pressures, as shown in figure 5.1. After the rupture of the diaphram, the pressure discontinuity propagates in the low pressure region, the expansion fan propagates in the high pressure region and the contact discontinuity separating the two gas regions propagates towards the low pressure region.

The computational grid points were equally distributed between the driver and the driven regions at time $t = 0$. The initial conditions were as follows:

$Driverside(leftside) : 10^6 Pa, T = 3000.0K, u = 0m/s$

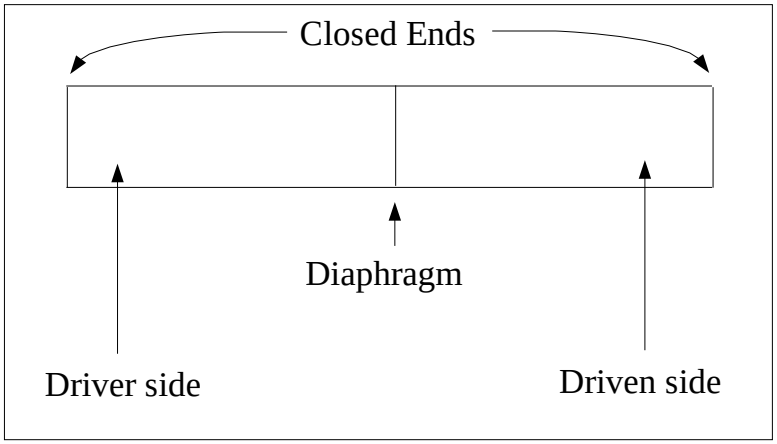

Figure 5.1: Shock Tube Problem

$Drivenside(rightside) : 10^5 Pa, T = 2200.0K, u = 0m/s$

In order to evaluate the case using the sub time steps, the driven side was set to be a slow time region, and the solution was advanced using sub time steps. Both the regions were assumed to contain the same gas. Newton iterations and 3 point backward time integration was used, with four Gauss-Seidel iterations. The results were compared with those of the original *Chem* code. The chemical models used were: ideal gas, air with five species and seventeen reactions, and a dissociating gas model [13]. The number of sub time steps used were two (i.e., $\delta t_1 = 2 * \delta t_2$). The performance of the present algorithm was compared to that of the original algorithm (not using the sub time steps). This was done by obtaining the time taken for the execution of the original code (with minimum time step, dtmin = 2.5e-08 and the total number of iterations as 3000), and comparing this value with present algorithm with $dtmin = 5e-08$ and the total number of iterations as 1500. The accurate local time stepping algorithm showed considerable speedup. Table 5.1 lists the different execution times of the code using the accurate local

Table 5.1: Performance Results on Shock tube simulations

Code	Chemistry Model	Time (sec)
ALT	Ideal gas	2103
Original	Ideal gas	2639
ALT	Air 5 species	6379
Original	Air 5 species	8521
ALT	Dissociating Oxygen	3029
Original	Dissociating Oxygen	4058

time stepping (ALT) and the one not using it (Original) for different chemistry models. The ALT method uses $dtmin = 5e-8$ and runs for 1500 iterations while the Original uses $dtmin = 2.5e-8$ and runs for 3000 iterations in all the cases. This shows that the ALT stepping is better for all cases, including those involving reactions of the species. A sample of the results is shown in figures 5.2 through 5.13. Notice that the solutions obtained with the two approaches are undistinguishable.

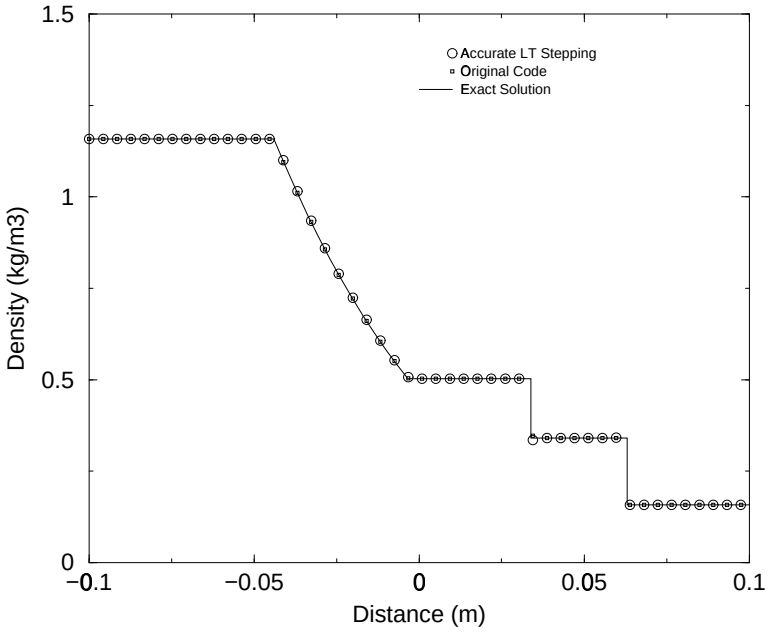

Figure 5.2: Ideal gas : Density Distribution

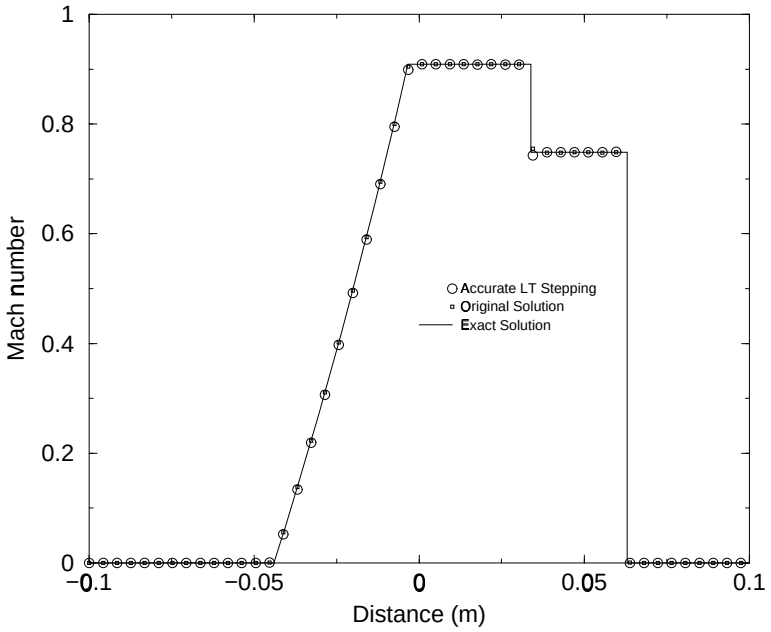

Figure 5.3: Ideal gas : Mach Number Distribution

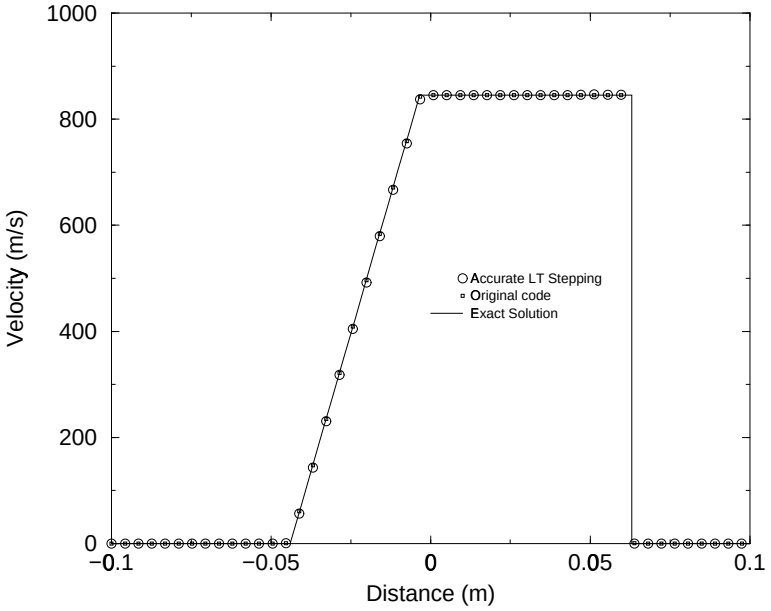

Figure 5.4: Ideal gas : Velocity Distribution

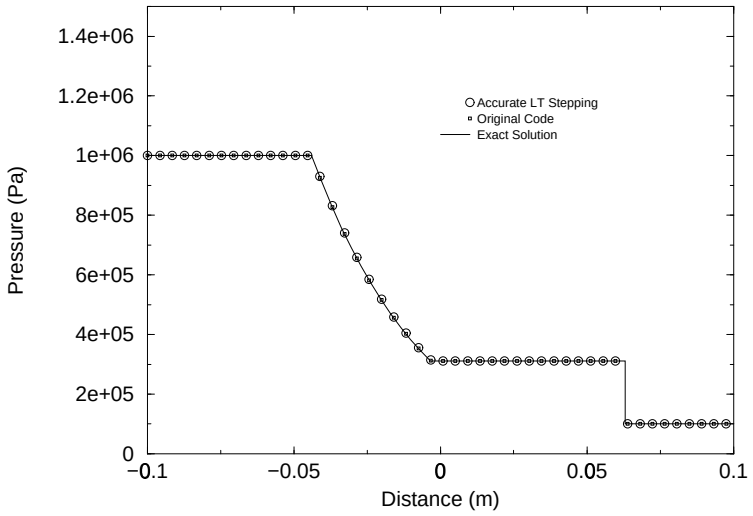

Figure 5.5: Ideal gas : Pressure Distribution

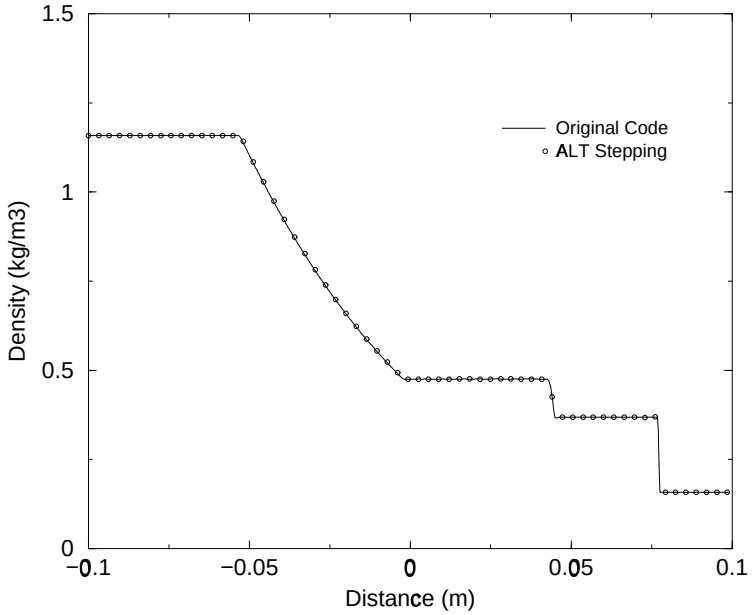

Figure 5.6: Air 5 species : Density Distribution

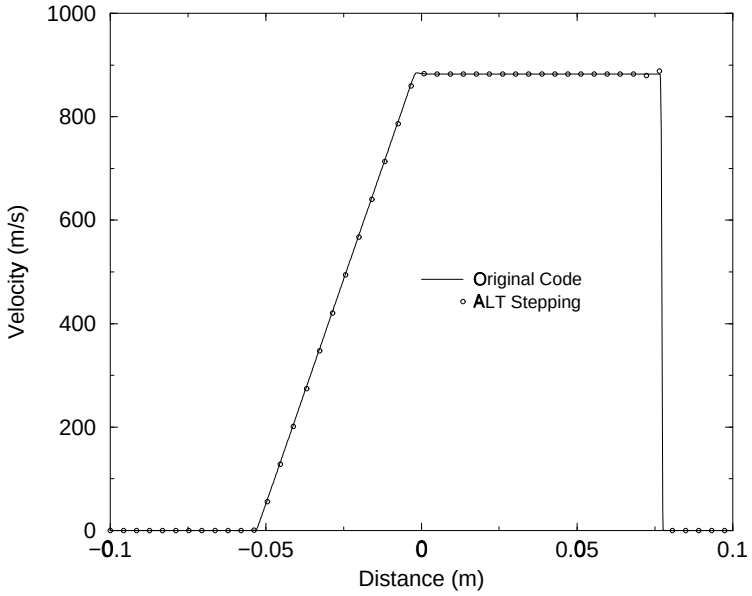

Figure 5.7: Air 5 species : Velocity Distribution

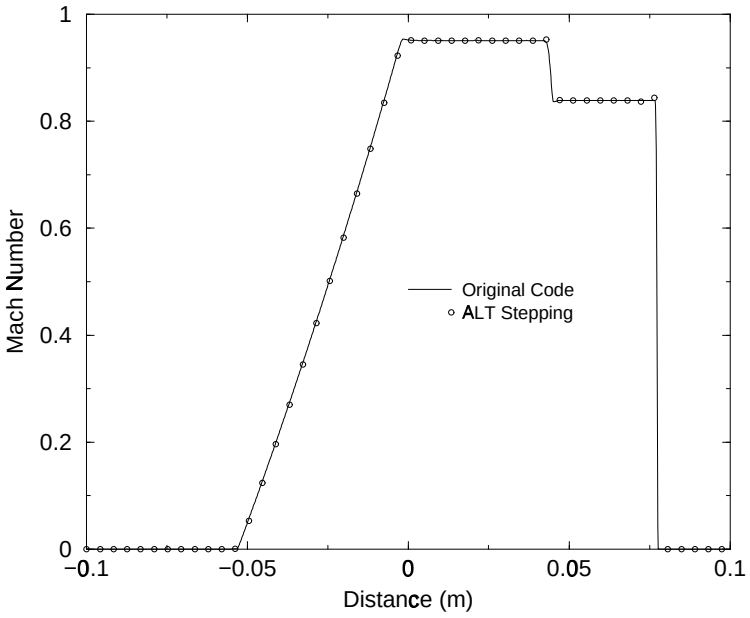

Figure 5.8: Air 5 species : Mach Number Distribution

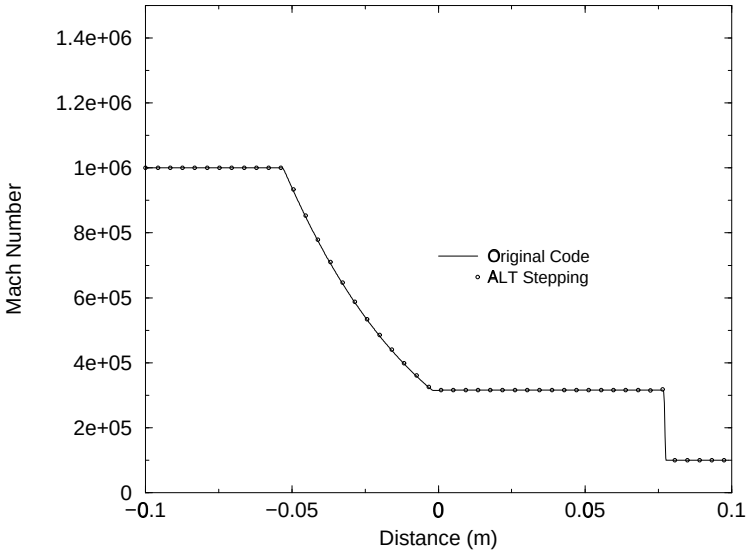

Figure 5.9: Air 5 species : Pressure Distribution

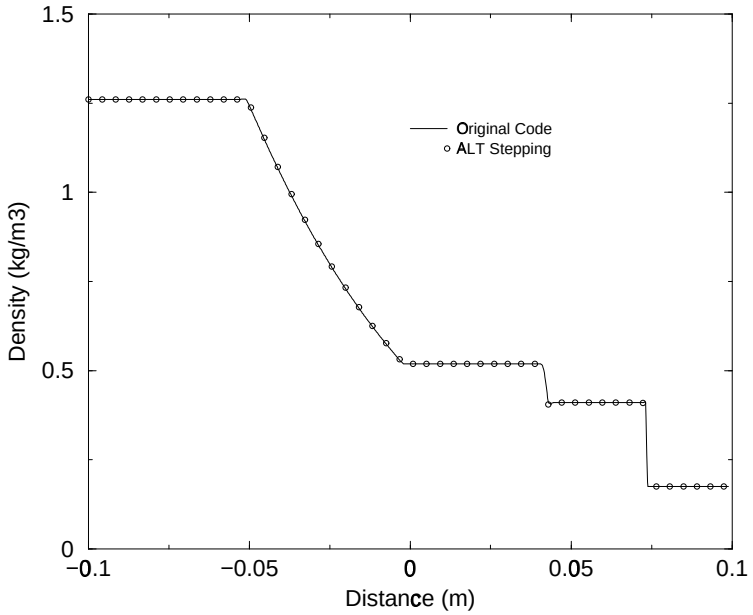

Figure 5.10: Dissociating Oxygen : Density Distribution

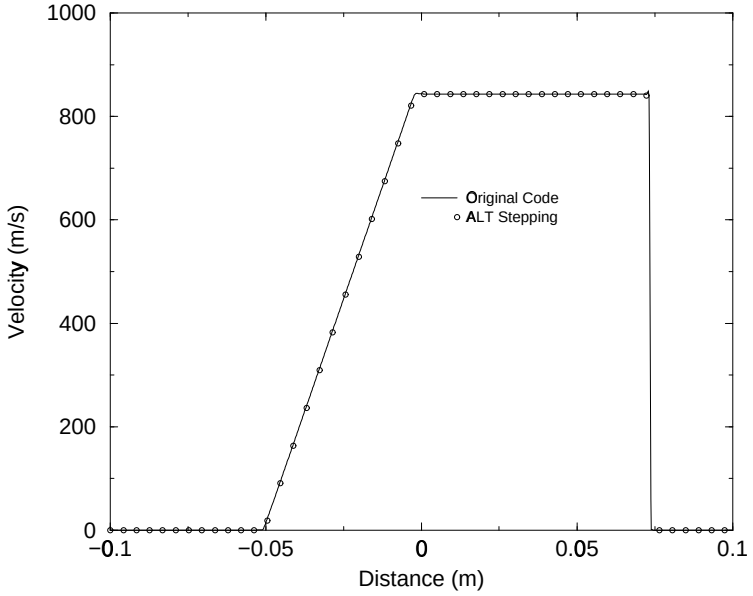

Figure 5.11: Dissociating Oxygen : Velocity Distribution

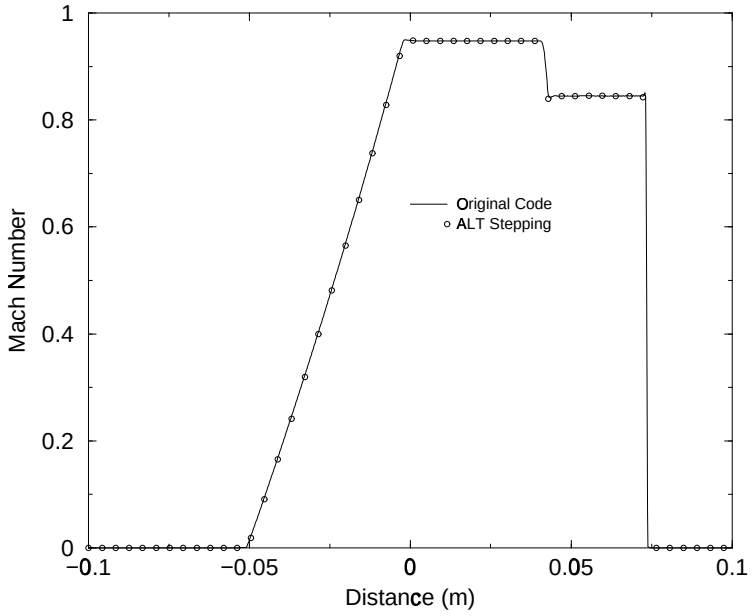

Figure 5.12: Dissociating Oxygen : Mach Number Distribution

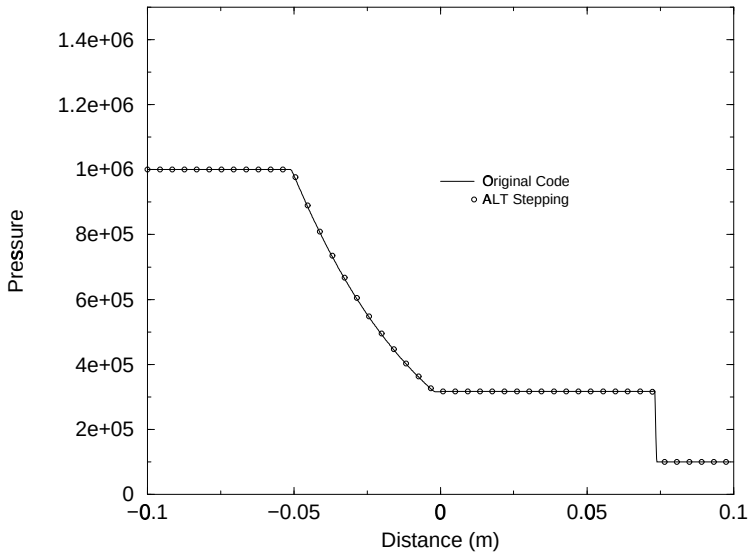

Figure 5.13: Dissociating Oxygen : Pressure Distribution

5.2 The Detonation Problem

A detonation wave can be regarded as consisting of an extremely strong shock wave closely followed by an exothermic reaction capable of providing enough energy to sustain the wave. It is far more violent and destructive than a shock wave due to the presence of a greater amount of energy produced from the reaction, and the chemical combustion is continuously initiated by the adiabatic compression and heating of the gaseous medium behind the shock front. Two of the main characteristics of a detonation are that it propagates with a steady constant velocity, and a sharp peak in pressure (called the Von Neumann spike) is observed at the detonation front. This is associated with the finite rate chemistry, which takes place after an extremely short induction time. High instantaneous over-pressures are therefore associated with detonations, particularly in the transition to detonation process.

The chemistry which occurs in an explosive material when an impinging shock wave initiates detonation is difficult to follow experimentally for two reasons: the time scale involved is very short, and the destructive nature of the detonation tends to destroy any nearby monitoring equipment. So, detonation waves are modelled using CFD techniques. Due to the sharp peak in pressure, near the detonation front, the solution needs to be very accurate and this can be achieved by using the technique of accurate local time stepping. The region around the detonation front can be subjected to sub time steps.

The detonation problem can be simulated by a setup as shown in figure 5.14.

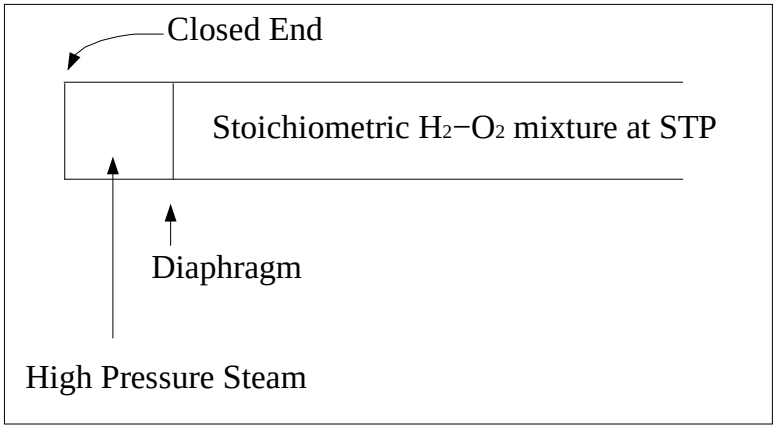

Figure 5.14: Detonation Problem

Table 5.2: Performance Results on Detonation Wave simulations

Code	Time (min)
ALT	464
Original	536

The initial conditions that were used for simulating the detonation problem are as under:

$Driverside(leftside) : 2 * 10^6 Pa, T = 3000.0K, u = 0m/s, mixture = [H2O = 1.0], equilibrium$

$Drivenside(rightside) : 1.2 * 10^5 Pa, T = 273.0K, u = 0m/s, mixture = [H2 = 0.125, O2 = 0.875]$

After the diaphragm breaks, the intense shock generated by the high pressure steam, initiates the detonation wave in the stoichiometric $H_2 - O_2$ mixture.

The results that were obtained from the accurate local time stepping were compared against the results from the original *Chem* code. The results were in good agreement with each other and are shown in figures 5.15 through 5.18.

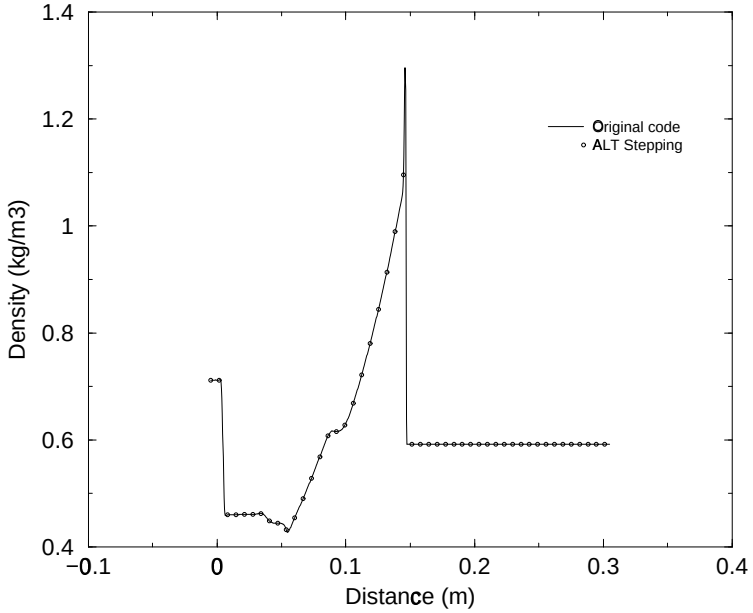

Figure 5.15: Density Distribution

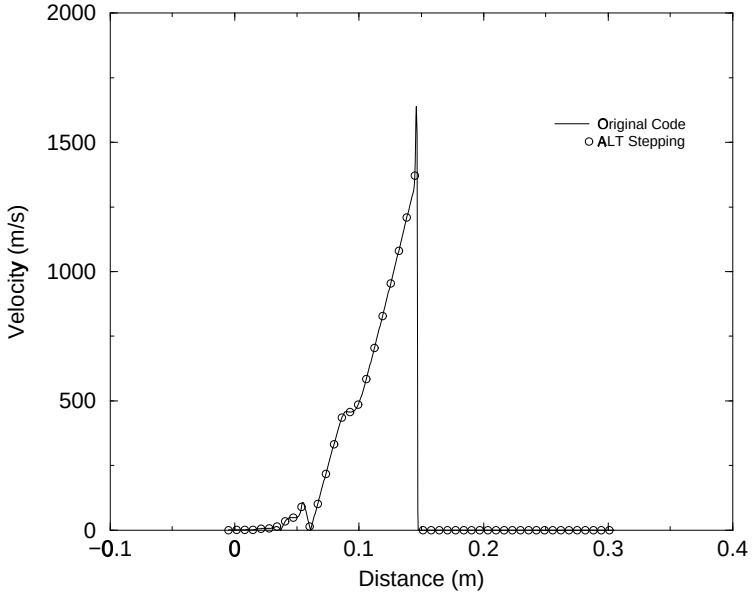

Figure 5.16: Velocity Distribution

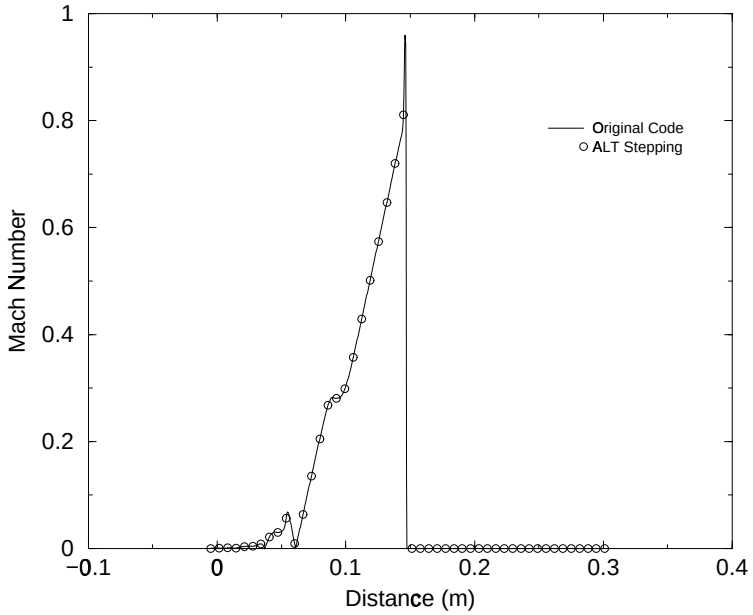

Figure 5.17: Mach Number Distribution

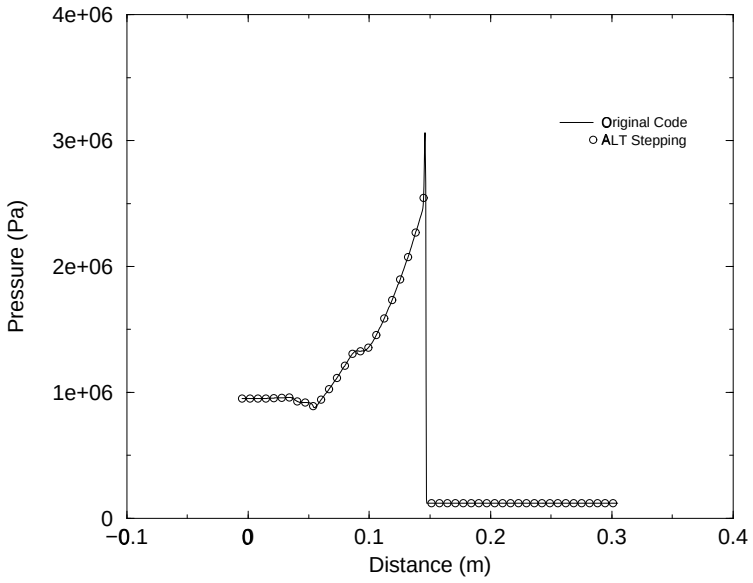

Figure 5.18: Pressure Distribution

CHAPTER VI

CONCLUSION AND FUTURE WORK

The present work was concerned with incorporating accurate local time stepping into the *Chem* code which would improve the performance of this flow solver. The improvement in performance was because the calculations were concentrated to a particular region (the slow time region) rather than the entire domain.

While this algorithm definitely improves the efficiency and accuracy of a flow solver, we need to be careful near the edges of the blocks. This is because the fluxes are not conserved at the edges of the blocks, due to which the error gets magnified as the time advances. Figure 6.1 shows the deviation of the flow solver from the original code. The same initial conditions mentioned in the previous chapter were used, but this time the wave was advanced (in time) so that it crosses the blocks. Three methods are suggested to overcome this problem.

- The first one takes advantage of the solution. The slow time region should be chosen such that the abrupt changes in the flow are well inside it and the gradual changes are in the fast time region. To explain this, we consider the shock tube problem. Here the fast time region consists of the expansion fan

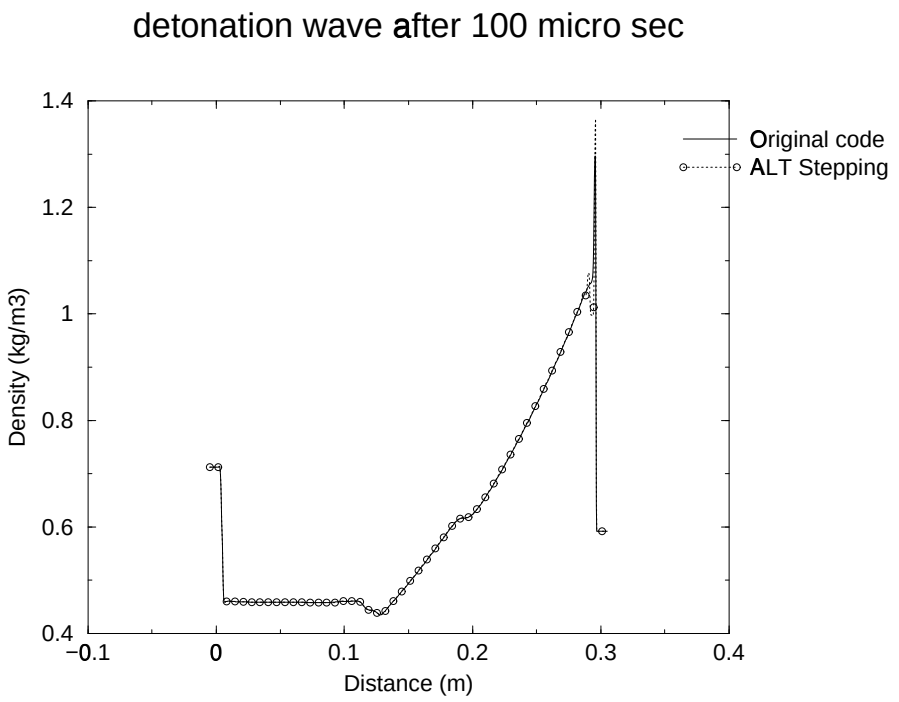

Figure 6.1: Conservation problem near the boundaries of the blocks

while the contact surface and the normal shock wave are in the slow time region. This is not a perfect solution because it is not generalizable.

- The second one is more generalized than the first one. In this method, the first few time steps are done using traditional approaches and when an abrupt change is detected in the solution, the region near it is chosen to be the slow time region, i.e., the algorithm adapts to the solution.

- The third method focusses on the fact that the discrepancies are due to the non-conservation of fluxes near the edges of the blocks. So, we could try to change the algorithm such that the fluxes are conserved even after the sub time steps near the edges of the blocks.

The items discussed above should be the topic of future work.

REFERENCES

[1] A. H. van den Boogaard, R. de Borst, and P. A. J. van den Bogert, "An adaptive time-stepping algorithm for quasistatic processes," *Communications in Numerical Methods in Engineering*, vol. 10, pp. 837–844, 1994.

[2] K. Chen, M. J. Baines, and P. K. Sweby, "On an adaptive time stepping strategy for solving nonlinear diffusion equations," *J. Computational Physics*, vol. 105, pp. 324–332, 1993.

[3] C. Hirsch, *Numerical Computation of Internal and External Flows: Volume 1 Fundamentals of Numerical Discretization*. John Wiley & Sons, 1988.

[4] P. Smolinski, "An explicit multi-time step integration method for second order equations," *Computer Methods in Applied Mechanics and Engineering*, vol. 94, pp. 25–34, 1992.

[5] V. Venkatakrishnan, "Implicit schemes and parallel computing in unstructured grid cfd," tech. rep., ICASE, 1995. ICASE Report No. 95-28.

[6] L. Cao and J. Zhu, "Accurate local time stepping algorithms for solving partial differential equations," *IMACS series in Computational and Applied Mathematics*, vol. 4, pp. 253–258, 1998.

[7] L. Cao and J. Zhu, "Efficient and accurate local time stepping algorithms for multi-rate problems," *Proceedings of the Eighth International Colloquium on Differential Equations*, vol. 4, pp. 97–104, 1998.

[8] P. Cinnella and B. Grossman, "Computational methods for chemically reacting flows," in *Handbook of Fluid Dynamics and Fluid Machinery* (J. A. Schetz and A. E. Fuhs, eds.), vol. 2, pp. 1541 – 1590, John Wiley & Sons, Inc., 1996.

[9] B. Gustafsson, "The convergence rate for difference approximations to mixed initial boundary value problems," *Mathematics of Computation*, vol. 29, pp. 396–406, 1975.

[10] O. Peroomian and S. Chakravarthy, "A feasibility study of a cell – averaged-based multi – dimensional ENO scheme for use in supersonic shear layers," tech. rep., AIAA, 1996. AIAA 96-0524.

[11] R. H. P. D. A. Anderson, J. C. Tennehill, *Computational Mechanics and Heat Transfer*. Hemisphere Publishing Co., McGraw-Hill Book Co., 1984.

[12] C. Cox, *An Efficient Solver for Flows with Non-Equilibrium*. PhD thesis, Mississippi State University, 1992.

[13] W. G. Vincenti and C. H. Kruger, *Introduction to Physical Gas Dynamics*. Krieger Publishing Company, 1965. Reprinted 1986.

[14] B. Mark, *Steps Towards More Accurate and Efficient Simulations of Reactive Flows*. PhD thesis, Mississippi State University, 1997.

[15] A. G. Hansen, "Generalized control volume analyses with application to the basic laws of mechanics and thermodynamics," *Bulletin of Mechanical Engineering Education*, vol. 4, pp. 161–168, 1965.

[16] J. Janus, *Advanced 3-D Algorithm for Turbomachinery*. PhD thesis, Mississippi State University, May 1989.

[17] E. A. Luke, *A Rule-based specification system for computational fluid dynamics*. PhD thesis, Mississippi State University, December 1999.

[18] E. Luke, "Loci: A deductive framework for graph-based algorithms," *ISCOPE*, pp. 142–153, 1999.